PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK UNTUK MEMANIPULASI MODEL DATA NETWORK

Disusun oleh :

ALRIYONO

NRP. 2689 100 033

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA

1996

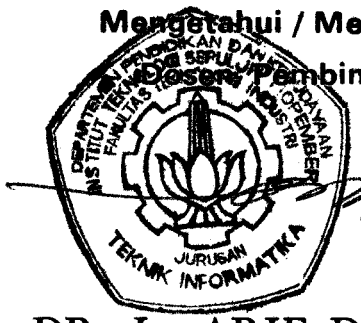
LEMBAR PENGESAHAN

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK UNTUK MEMANIPULASI MODEL DATA NETWORK

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik Informatika
Pada
Jurusan Teknik Informatika
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
S u r a b a y a

Mengetahui / Menyetujui
Dosen Pembimbing



DR. Ir. ARIF DJUNAIDY

NIP. 131 633 403

S U R A B A Y A
September, 1996

Abstrak

Hal yang paling penting pada pemodelan data adalah menyiapkan sebuah model yang dapat digunakan dalam pemrosesan data. Ada tiga metode dalam pemodelan data yaitu model data hirarki, model data *network* dan model data *relasional*. Pada model data hirarki data disusun dalam struktur *tree*. Model data *network* menggunakan *graph* untuk menggambarkan relasi data. Model data *relational* menggunakan tabel dan konsep matematika untuk manipulasi data.

Pada Tugas Akhir ini dirancang dan dibuat perangkat lunak yang dapat membangun serta memanipulasi model data *network*. Untuk manipulasi dibuat translator sederhana dengan menggabungkan banyak perintah . Pada perangkat lunak ini juga akan diketahui hasil pemakaian aturan penyisipan (*insertion constraint*), aturan keberadaan record dalam database(*retention constraint*) , aturan urutan record dalam *set occurence* serta aturan seleksi set.

KATA PENGANTAR

Alhamdulillahirobbil Alamin, segala puji bagi Allah SWT, karena dengan hidayah dan ridloNya penulis dapat menyelesaikan pembuatan Tugas Akhir yang berjudul :

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK UNTUK MEMANIPULASI MODEL DATA NETWORK

Tugas akhir ini dikerjakan untuk memenuhi syarat akademis dalam rangka ujian akhir mahasiswa Strata 1 Jurusan Teknik Informatika FTI ITS Surabaya.

Penulis menyadari bahwa dalam menyelesaikan Tugas Akhir ini tidak terlepas dari bantuan pihak lain, maka dalam kesempatan ini penulis mengucapkan terima kasih yang sebesar-besarnya , terutama kepada :

1. Dr. Ir. Handayani Tjandrasa, MSc selaku Ketua Jurusan Teknik Informatika FTI ITS.
2. Dr. Ir. Arief Djunaidy, MSc selaku dosen pembimbing dan dosen wali yang telah membimbing dan mengarahkan penulis selama menyelesaikan tugas akhir.
3. Bapak, ibu, kakak dan adik yang telah banyak memberikan dorongan dan semangat.

4. Seluruh staf dosen Jurusan Teknik Informatika FTI ITS yang telah memberikan materi kuliah yang bermanfaat dalam penyelesaian Tugas Akhir ini.
5. Seluruh karyawan Jurusan Teknik Informatika FTI ITS yang membantu menyelesaikan administrasi kemahasiswaan.
6. Semua teman Angkatan C05 terutama Arief, Heru, Ibud, Thoyibah, Jamak, Alik,Arfan,Widjoyono yang selalu bersama-sama bertekad segera lulus.

Penulis menyadari bahwa masih banyak kekurangan dan kelemahan dalam menyelesaikan tugas akhir ini, karena itu kritik dan saran yang tambah menyempurnakan sangat diharapkan.

Akhirnya penulis berharap Tugas akhir ini dapat bermanfaat bagi kita semua khususnya bagi rekan-rekan mahasiswa dan masyarakat pada umumnya.

Surabaya , September 1996

Penulis

DAFTAR ISI

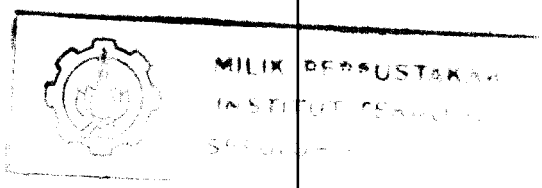
Halaman

DAFTAR GAMBAR

Halaman

BAB I

PENDAHULUAN



1.1 Latar belakang

Pemodelan data adalah salah satu tahap yang terpenting dalam pengembangan sistem informasi dalam komputer. Sebelum membuat *Database Management System* kita harus mendefinisikan data yang dimanipulasi dan *relationship* antara data secara tepat dan akurat. Hal ini terjadi sebagai konsekuensi analisa kejadian alamiah yang dinamis dan komplek. Sebuah model data merupakan representasi dari kejadian alamiah dimana aspek yang relevan dengan kebutuhan terdapat didalamnya . Model data adalah proses mengorganisasi struktur data dalam bentuk yang siap diimplementasikan pada komputer. Sehingga kita menyatukan proses menyusun, mengumpulkan dan memilah data dalam unit yang siap dimanipulasi oleh komputer. Terdapat beberapa metoda dalam model data, dimana menurut urutan waktu pengembangannya adalah metode hirarki, *network* dan *relational*. Metode hirarki dan *network* dikembangkan berdasarkan pengetahuan mengenai penyimpanan data dalam komputer. Sedangkan pengetahuan mengenai matematika digunakan untuk pengembangan metode *relational.*

Dalam Tugas Akhir ini dirancang dan dibuat suatu perangkat lunak yang mampu memanipulasi model data *network*. Harapan penulis perangkat lunak ini

dapat menambah pengetahuan mengenai apa dan bagaimana struktur data pada model data *network* dan memanipulasinya.

1.2. Tujuan

Tujuan pokok Tugas Akhir ini adalah merancang dan membuat perangkat lunak yang mampu memanipulasi model data *network*.

1.3. Permasalahan

Untuk memenuhi tujuan yang tersebut diatas, terdapat beberapa permasalahan :

-Implementasi penyimpanan model data *network* yang mendukung proses manipulasinya.

-Implementasi perangkat lunak yang mampu memanipulasi model data *network* yang meliputi *design sistem,* diagram aliran datadan diskripsi fungsi.

1.4. Pembatasan Masalah

Pada Tugas Akhir ini kemampuan perangkat lunak adalah memanipulasi model data *network* dengan perintah *navigasi, retrieval* , *update* serta mematuhi *contraint databasenya* dan menunjukkan *current indikator* baik record dan set setiap perintah yang dikerjakan.

1.5. Sistematika Pembahasan

Sistematika pembahasan konsep Tugas Akhir ini adalah sebagai berikut:

BAB II membahas dasar teori model data network. Akan dijelaskan konsep *binary relasionship*, teori model data *network* dan manipulasi, serta implementasi fisik model data *network*.

BAB III membahas perancangan sistem perangkat lunak yang meliputi design sistem, diagram aliran data dan diskripsi fungsi untuk memanipulasi model data *network*.

BAB IV membahas pembuatan perangkat lunak meliputi penjelasan struktur data , fungsi dan prosedur yang dipakai dalam program.

BAB V membahas uji coba dan evaluasi perangkat lunak.

BAB VI merupakan bab penutup membahas kesimpulan dan saran yang berperan untuk pengembangan selanjutnya.

BAB II

DASAR TEORI MODEL DATA NETWORK

2.1. Model Data *Network*

Model data *network* adalah "model data yang terdiri dari record-record yang diorganisasi dalam bentuk *graph*".[1] Dalam model data *network* struktur datanya terdiri dari record dan set. Model data *network* hanya dapat menyelesaikan relasi 1:1 dan 1:M.

2.1.1. Record , Tipe Record dan Item Data

Record adalah tempat menyimpan data. Tipe record adalah sekelompok record yang menyimpan tipe informasi yang sama. Setiap tipe record terdiri dari item data yang mempunyai nama dan format tertentu.

2.1.2. Tipe Set

Tipe set adalah "gambaran (*deskripsi)* relasi 1: N antara dua tipe record , dimana setiap tipe set terdapat tiga elemen utama yaitu nama tipe set , tipe record *owner* dan record tipe *member*." [2] Dalam sebuah tipe set terdiri dari banyak *set*

---

[1] Henry F Korth Abraham, Silber Schatz, Database System Concepts , McGraw-Hill , New York , 1986 , halaman 107

[2] Ramez Elmasri And B Navanthe, Fundamental Of Database System , The Benjamin/Cumming , 1989 , halaman 289

occurence atau *set instance*. Untuk setiap *set instance* terdiri dari satu record dari tipe record *owner* dan beberapa(kosong) record dari tipe record *member*.

Sebuah record dari *member* tipe record tidak dapat berada pada lebih dari satu *set occurence*. Hal ini sesuai dengan relasi yang dapat diterapkan pada model data *network* yaitu 1:M. Pada model data *network* untuk menyelesaikan relasi M:N digunakan dua tipe set dan tipe record *dummy*. Sebuah *set occurence* digambarkan dengan *ring linking* antara record *owner* dan record *member*.

Perbedaan antara *set occurence/instance* dan *set* dalam disiplin ilmu matematika adalah :

- *set instance* mempunyai elemen yang berbeda yaitu record dari tipe record *owner* dan record dari tipe record *member*. Tetapi *set* dalam matematika tidak dibedakan.
- record-record *member* dari sebuah *set instance* diatur urutannya, sedangkan dalam matematika elemen set tidak diatur urutannya.

Pada model data *network* terdapat tipe set khusus yaitu:

a. *System owned set* [1]

Tipe set ini tidak mempunyai tipe record *owner*, *system* sebagai owner. Tipe set ini disebut *singular set* karena hanya terdapat *satu set occurence*. Ada dua manfaat utama dari tipe set ini yaitu :

1.sebagai *entry point* ke database melalui record *member* tertentu.

2.digunakan untuk mengurutkan record-record dari suatu tipe record dengan menggunakan aturan urutan tertentu.

---

[1] ibid, halaman 292

b. *Multimember set* [2]

Pada tipe set ini record member dari sebuah *set occurence* berasal dari beberapa tipe record.

c. *Recursive set* [3]

Pada *recursive set* ini, sebuah record dapat sebagai *owner* dari suatu *set occurence* serta dapat sebagai *member* pada *set occurence* lainnya dalam tipe set yang sama. Tipe set ini dihindarkan karena sukar untuk dimanipulasi.

2.1.3. Aturan(*constraint*) Model Data *Network*

2.1.3.1. Aturan Penyisipan(*insertion constraint*) pada Tipe Set

Aturan ini berlaku ketika menyisipkan sebuah record pada database dan record baru tersebut berpartisipasi sebagai *member*. Ada dua pilihan dalam penyisipan ini yaitu :

a. Automatic [4]

Sebuah record *member* baru secara otomatis direlasikan ke suatu *set occurence* yang tepat berdasarkan syarat pada deklarasi *SET SELECTION* .

b. Manual [5]

Sebuah record *member* baru direlasikan pada suatu *set occurence* setelah dilakukan perintah CONNECT.

---

[2] ibid.
[3] ibid.
[4] ibid, halaman 299.
[5] ibid.

2.1.3.2. Retention Constraint

Aturan ini menentukan bilamana sebuah record dari tipe record *member* harus direlasikan ke suatu *set occurence* atau boleh bebas tanpa direlasikan. Ada tiga jenis aturan *retention* ini yaitu :

a. Optional [6]

Record dari tipe record *member* boleh berada dalam database tanpa sebagai *member* dari salah satu *set occurence.*

b. Mandatory [7]

Semua record dari tipe record *member* harus direlasikan pada salah satu *set occurence*, tetapi record dapat berpindah diantara *set occurence* dengan perintah RECONNECT.

c. Fixed [8]

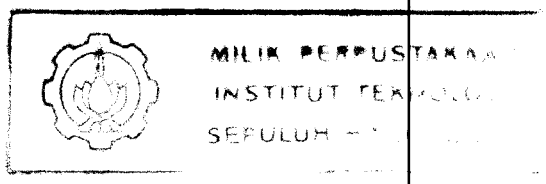
Sama seperti *mandatory*, tetapi record tidak dapat berpindah diantara *set occurence.*

---

[6] ibid.
[7] ibid , halaman 300
[8] ibid.

2.1.3.3. Aturan Menyusun Set (*Set ordering constraint*)

Record-record *member* dalam sebuah *set occurence* dapat disusun berdasar aturan sebagai berikut :



a. Sorted [9]

Record-record diurutkan berdasarkan nilai *field* kunci dari tipe record *member.*

b. First atau Last [10]

Bilamana *first* maka record baru disisipkan tepat setelah record *owner*, sebaliknya bila *last* maka record baru tersebut disisipkan paling akhir dari urutan record *member* yang ada pada *set occurence* tersebut.

c. Next atau Prior [11]

Bilamana *next* maka record baru disisipkan setelah record terakhir yang dimanipulasi pada *set occurence* tersebut. Untuk *prior* disisipkan sebelumnya.

2.2. Definisi Data pada Model Data *Network*

Definisi data ini untuk mendeklarasikan tipe record, tipe set, definisi item data dan *constraint* skema.

2.2.1. Deklarasi Tipe Record dan Item Data

Deklarasi tipe record dan item data ini terdiri dari nama tipe record, nama item data dan formatnya. Untuk mencegah penggandakan (duplikasi) data maka pada deklarasi tipe record ditambahkan nama item data sebagai kuncinya.

---

[9] ibid , halaman 302
[10] ibid.
[11] ibid.

2.2.2. Deklarasi Tipe Set dan Aturan Seleksi Set

Deklarasi tipe set lebih banyak aturan yang harus dinyatakan yaitu nama tipe set, nama *owner, key owner, order,* nama *member, key member, insertion, retention , selection.*

Bila aturan penyisipan (*insertion constraint*) adalah AUTOMATIC, maka sistem harus mencari *set occurence* sebagai tujuan dengan memperhatikan "aturan seleksi(*selection constraint*) " [12]yang terdiri dari :

a. *SET SELECTION IS STRUKTURAL field record owner = field record member.*

b. *SET SELECTION IS APPLICATION*

Set occurence sebagai tujuan harus sebagai tipe set yang terakhir dimanipulasi.

c. *SET SELECTION IS BY VALUE OF* <nama_field> *IN* <nama_tipe_record>

Tujuan *set occurence* ditentukan dengan mengeset nilai field di *unit work area.*

[12] ibid , halaman 306

2.3. Konsep Dasar Untuk Manipulasi Database *Network*

Pada sub-bab ini akan dibahas konsep dasar program memanipulasi model data *network(DML)*. Perintah-perintah DML pada model data ini adalah **"Record at a time"** [13] artinya pada satu waktu hanya satu record yang dapat dimanipulasi. Oleh karena itu peran *currency indikator* sangat penting. *"Currency indikator "* [14] terdiri dari:

a. Current of record type

Current indikator ini menunjukkan tipe record yang sedang atau telah diakses oleh DBMS. Jika tak ada record yang sedang atau telah diakses maka *current* indikator tipe record tidak didefinisikan.

b. Current of set type

Current indikator ini menunjukkan *set occurence* yang sedang atau telah diakses oleh DBMS. Untuk sebuah *set occurence*, maka *current* indikator tipe set ditunjukkan oleh sebuah *owner* atau *member* recordnya. Jika program tidak pernah mengakses record dari set tipe, maka *current* indikator tipe setnya tidak didefinisikan.

c. Current of run unit(CRU)

Current indikator ini menunjukkan tipe record sedang berada di *user work area (UWA)*.

---

[13] ibid , halaman 311
[14] ibid , halaman 312

Perintah untuk memanipulasi sebuah database *network* disebut "*Network DML* dikelompokkan menjadi 3 " [15]yaitu:

a. Perintah Navigasi(Navigasi commands)

Perintah ini digunakan untuk menentukan *currency indikator* untuk record dan *set occurence* tertentu dari database. Untuk *navigasi* dilakukan dengan perintah FIND.

b. Perintah Retrieval(Retrieval commands)

Perintah ini digunakan untuk mendapatkan *(meretrieve) current record* dari CRU. Perintah GET untuk meletakkan record dari CRU pada UWA.

c. Perintah Update(Update commands)

Perintah ini dibagi dua yaitu:

-Update record

menyimpan record baru dilakukan perintah STORE, menghapus record dilakukan perintah ERASE, merubah nilai *fields* dengan perintah MODIFY.

-Update set occurences

untuk memasukkan sebuah record *member* dalam sebuah *set occurence* dilakukan dengan perintah CONNECT Sebaliknya dengan perintah DISCONNECT untuk memisahkannya. Perintah RECONNECT untuk memisahkan sekaligus memasukkan sebuah record *member* dari *set occurence* satu ke *set occurence* yang lain.

---

[15] ibid , halaman 316

2.3.1. Format Perintah

2.3.1.1. Perintah Navigasi(Navigation commands)

-Perintah FIND

Perintah ini terdiri dari :

-1. DML untuk sebuah tipe record

Formatnya: "FIND ANY <nama record> [USING <list fields>]" [16]

Kegunaan: Menemukan record yang pertama kali memenuhi syarat yang ditentukan dalam USING .

Formatnya: "FIND DUPLICATE <nama record> [USING<list fields>] " [17]

Kegunaan: Menemukan record selanjutnya dari *current* indikator tipe record yang memenuhi syarat yang ditentukan dalam USING.

-2. DML untuk set occurence

Formatnya: "FIND (FIRST | NEXT | PRIOR | LAST| ...) <nama record> WITHIN <nama set> [USING<list fields>]"[18]

Kegunaan: Menemukan record yang berpartisipasi dalam sebuah *set occurence.*

---

[16] ibid , halaman 317
[17] ibid.
[18] ibid , halaman 319

Formatnya: "FIND OWNER WITHIN <nama set>" [19]

Kegunaan: Menemukan *owner* dalam sebuah *set occurence.*

Akibat dari perintah FIND diatas adalah terjadinya perubahan pada *current* indikator baik tipe record, tipe set dan kontrol unit..

2.3.1.2. Perintah Retrieval(retrieval command)

Formatnya: "GET <nama record> " [20]

Kegunaan: Menampilkan record pada *current control unit.*

2.3.1.3. Perintah *Update(Update commands)*

Perintah ini terdiri dari :

-a. *Update* record

Formatnya: "STORE <nama record>" [21]

Kegunaan: Menyisipkan sebuah record baru di *current* indikator tipe record ke dalam database.

Jika sebuah set bertipe *automatic* maka record baru harus otomatis disisipkan atau dihubungkan dan ini ditentukan dengan SET SELECTION.

Formatnya:"ERASE <nama record>" [22]

Kegunaan: Menghapus sebuah record yang terdapat pada *current* indikator tipe record. Penghapusan ini berpengaruh pada

---

[19] ibid.
[20] ibid , halaman 318
[21] ibid , halaman 322
[22] ibid , halaman 324

record-record *member* dimana record yang dihapus sebagai *ownernya* dan bergantung pada *retention* dari tipe set tersebut.

Formatnya: "MODIFY <nama record>" [23]

Kegunaan: Merubah beberapa nilai field dari record yang terdapat pada *current* indikator tipe record.

-b. Update set instance

Formatnya: "CONNECT <nama record>" [24]

Kegunaan: Menghubungkan record *member* di *current record indikator* pada *set instance* di *current* indikator tipe set . Perintah ini dapat dilakukan bila record *member* belum terhubung dengan record *owner* lainnya pada set sama dan tipe set adalah manual atau *automatic optional.*

Formatnya: "DISCONNECT <nama record>" [25]

Kegunaan: Memutuskan hubungan record member di *current record indikator* dari *set occur*ence tanpa menghubungkan ke *set occurence* lainnya. Perintah ini hanya dilakukan pada set bertipe optional.

Formatnya: "RECONNECT <nama record> WITHIN <nama set>" [26]

Kegunaan: Memindahkan record *member* dari satu *set occurence* ke *set occurence* yang lain. Perintah ini hanya dapat dilakukan pada set bertipe *optional dan mandatory.*

---

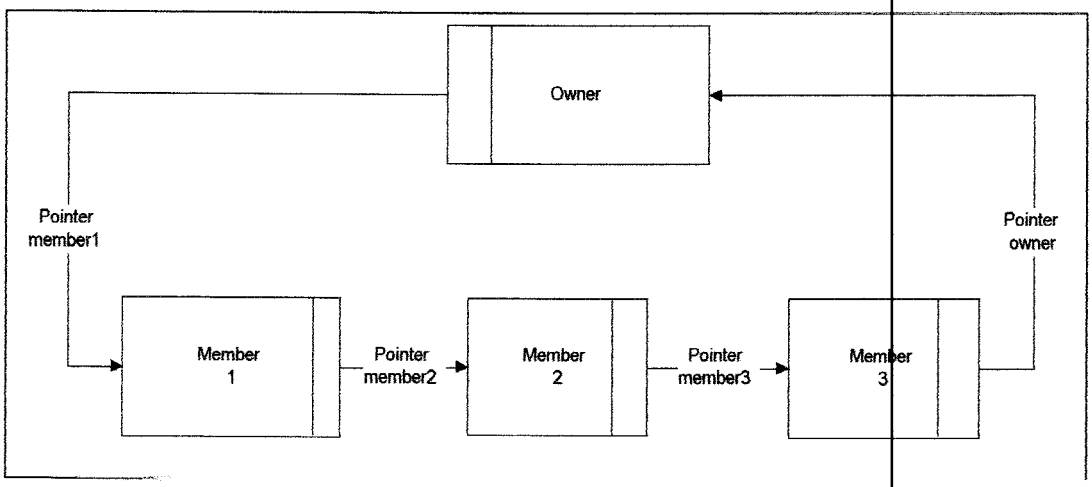
[23] ibid , halaman 326
[24] ibid.
[25] ibid.
[26] ibid.

2.4. Implementasi relasi

Untuk mengimplementasikan relasi dalam model data network ada dua cara yaitu :

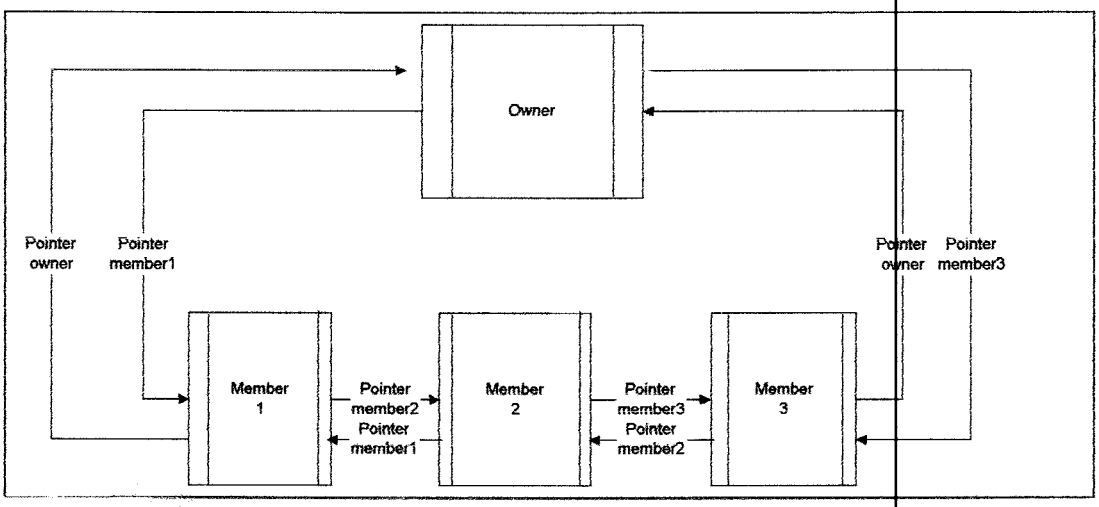
a. Link list

Disini dibentuk satu record *owner* dan beberapa record *member* seperti gambar 2.1.



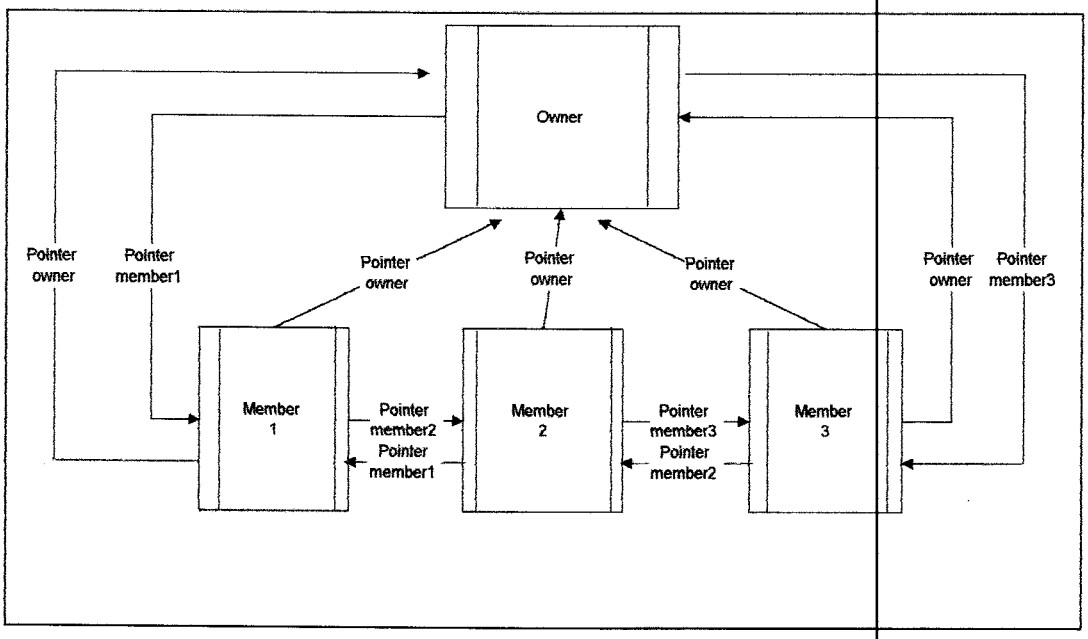
Gambar 2.1 Link list

Pada gambar di atas record *owner* mempunyai penunjuk alamat ke record *member* 1, record *member* ini mempunyai penunjuk alamat ke record *member* 2 dan selanjutnya sampai record *member* terakhir mempunyai penunjuk alamat ke record *owner* sehingga membentuk rantai tertutup (*a closed chain*). Proses penyisipan pada struktur ini sederhana tetapi jika *member* baru disisipkan berdasarkan urutan waktu tidak efisien. Karena harus menelusuri seluruh *member* agar menemukan record *member* terakhir. Untuk memperbaiki dibuatlah struktur seperti gambar 2.2.



Gambar 2.2 Double link list

Pada gambar 2.2 di atas, struktur tidak efisien bila ada record *member* yang berpartisipasi pada tipe set yang berbeda karena untuk memanipulasi dari record *owner* ke satu menuju ke record *owner* ke dua harus menelusuri semua record *member*. Untuk memperbaiki dibuat struktur seperti gambar 2.3.

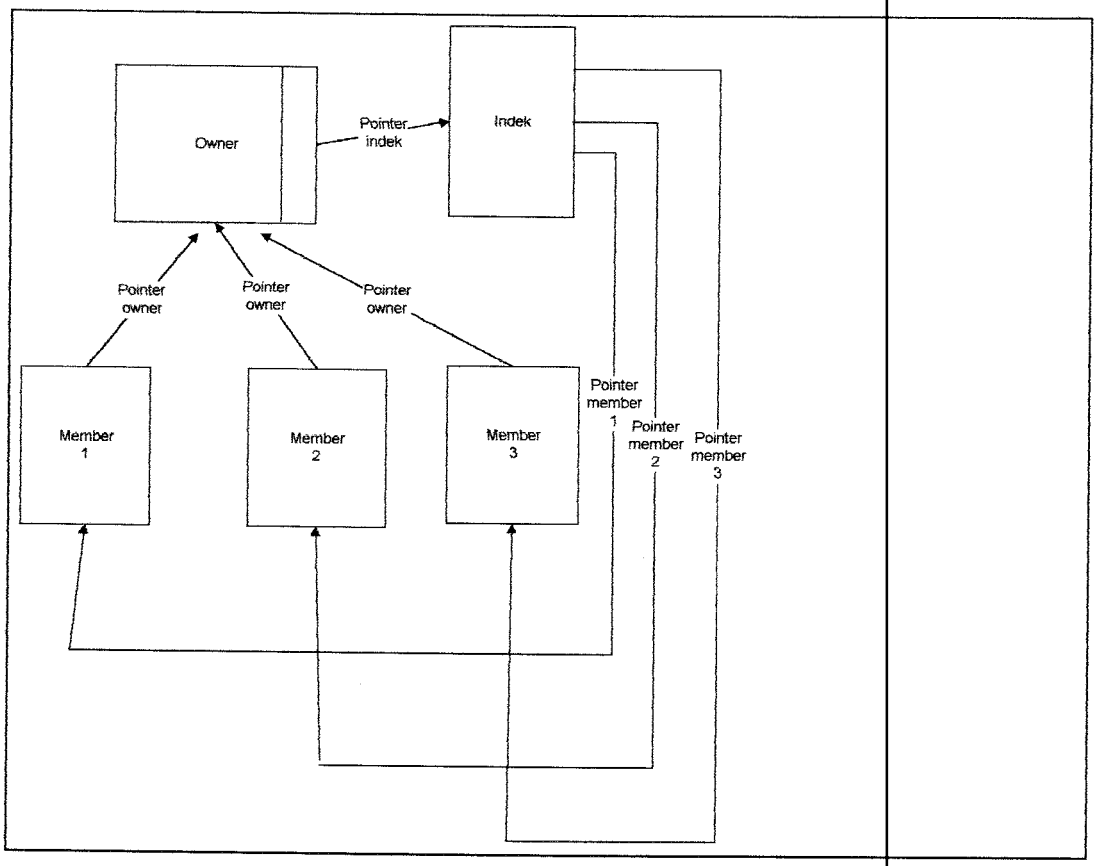


Gambar 2.3 Double link list dengan pointer owner

Pada struktur terakhir tempat penyimpanan yang diperlukan lebih banyak sehingga harus bijaksana dalam menentukan keseimbangan antara biaya penyimpanan dan efisiennya.

2. Indek

Struktur indek dapat digambarkan seperti gambar 2.4. Untuk mengakses record *member* dari record *owner* , indek yang berisi tabel alamat harus diakses terlebih dahulu. Keuntungan dalam struktur ini adalah record *member* hanya mempunyai penunjuk alamat ke record *owner*, penyimpanan alamat terpusat di dalam indek. Jika record *member* secara fisik berpindah maka yang diperbaiki hanya tabel indek.



Gambar 2.4 Indek

BAB III

PERANCANGAN SISTEM

Bab ini membahas perancangan dari perangkat lunak secara garis besar. Tujuan dari sistem yang dibuat adalah membuat lingkungan dimana pemakai dapat memasukkan perintah melalui *keyboard* atau menggabungkan beberapa perintah dalam file teks. Tujuan lainnya yaitu menghasilkan keluaran sesuai dengan perintah yang dikerjakan dan menampilkan perubahan pad *current* indikator tipe record, tipe set dan kontrol unit. Pada sub-bab ini akan dijelaskan proses perancangan perangkat lunak dengan diagram aliran data kemudian diskripsi fungsi atau algoritma fungsi.

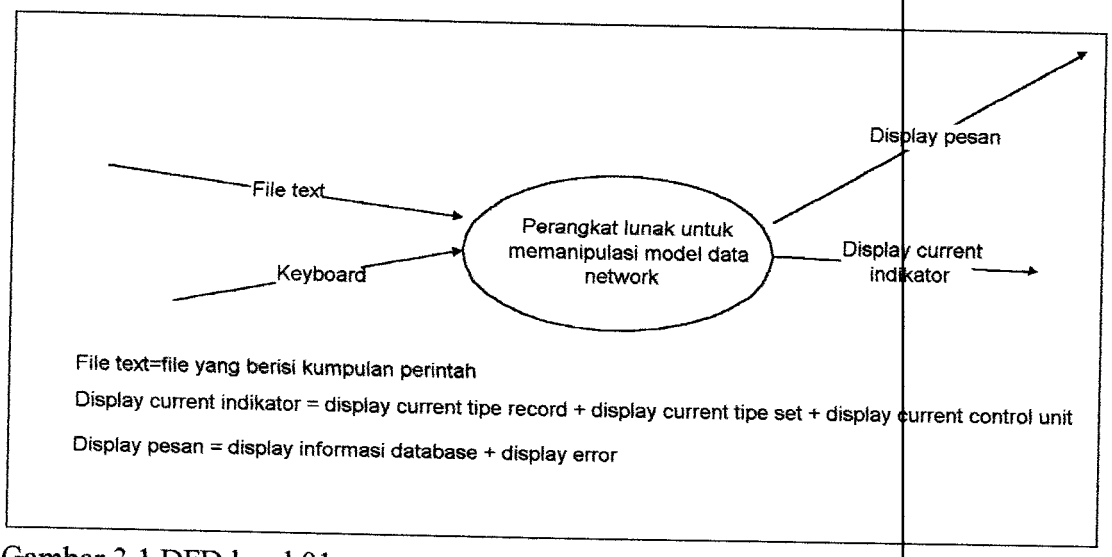
Diagram aliran data adalah "teknik grafik untuk menggambarkan aliran informasi dari proses-proses yang dilakukan perangkat lunak."[27]Pada level satu diagram aliran data disebut dengan fundamental system model dimana seluruh elemen perangkat lunak digambarkan dalam satu proses.

Diskripsi atau algoritma fungsi menjelaskan proses-proses yang digambarkan dalam diagram aliran data dengan tanpa memperhatikan konsep dasar prosedur yaitu *sequence*, seleksi dan perulangan.

---

[27] Roger S Pressman Ph. D, Software Engineering, Second edition, McGraw-Hill, New York, 1987

3.1 Diagram Aliran Data

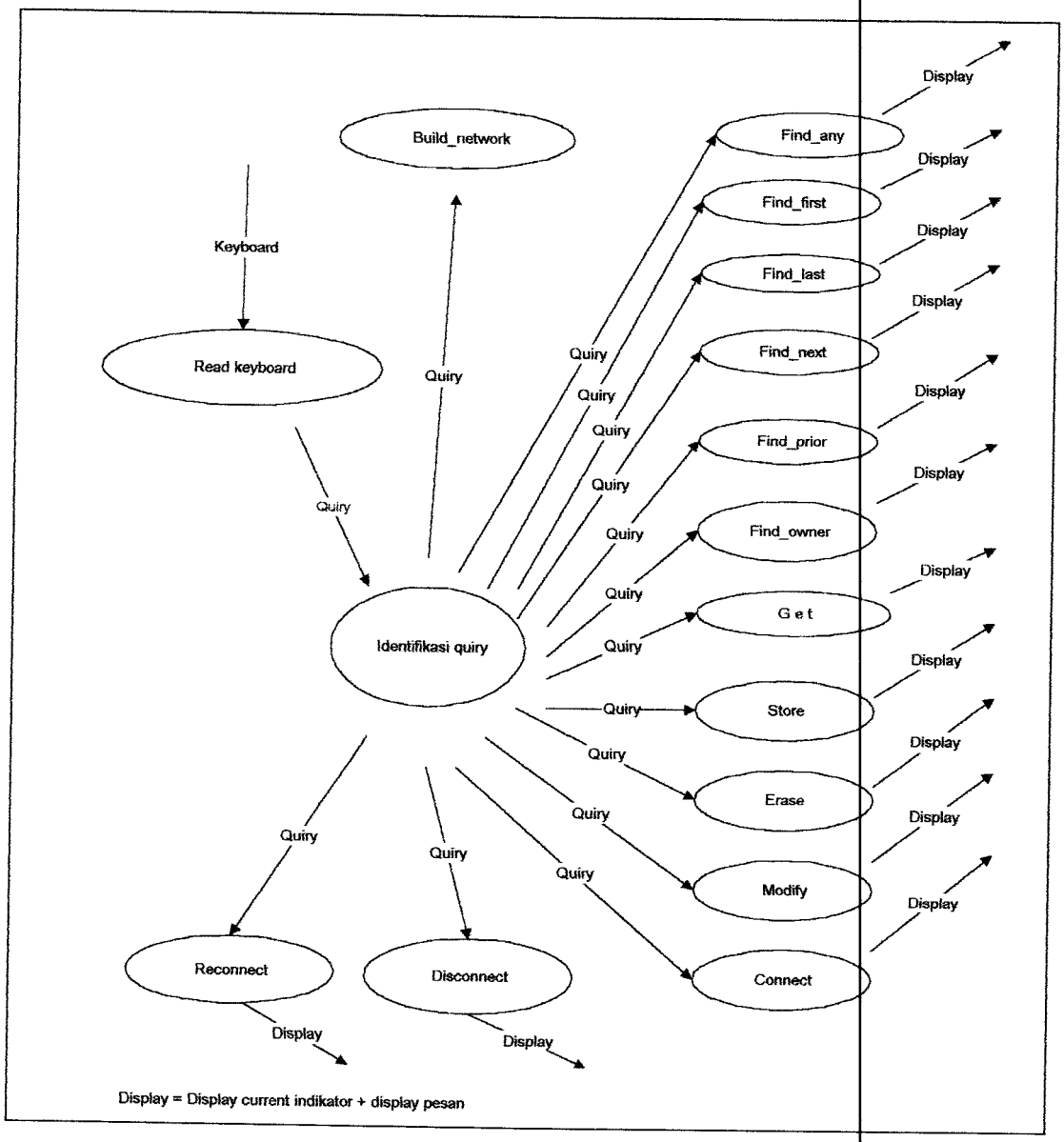


Gambar 3.1 DFD level 01

3.1.1 Diagram Aliran Data Level 01

Pada diagram aliran data level 01 diatas perangkat lunak dirancang dan dibuat untuk memanipulasi model data network . Input perangkat lunak yaitu *keyboard* dan file teks. *Keyboard* untuk memasukkan perintah manipulasi model data *network.* File teks terdiri beberapa perintah yang dapat dikombinasikan dengan uji kondisi IF atau looping WHILE. File ini dapat diedit dengan EDIT.EXE pada DOS. *Output* perangkat lunak adalah *display current* indikator dan *display* pesan. *Display current* indikator menampilkan *current* indikator untuk tipe record, tipe set dan kontrol unit. *Display* ini sangat penting karena perintah dalam model data *network* adalah *record at a time. Display* pesan menampilkan informasi *database* yang diinginkan dan pesan kesalahan yang diakibatkan oleh input perintah yang salah, *contraint database* yang tidak ditaati.

3.1.2. Diagram Aliran Data Level 02

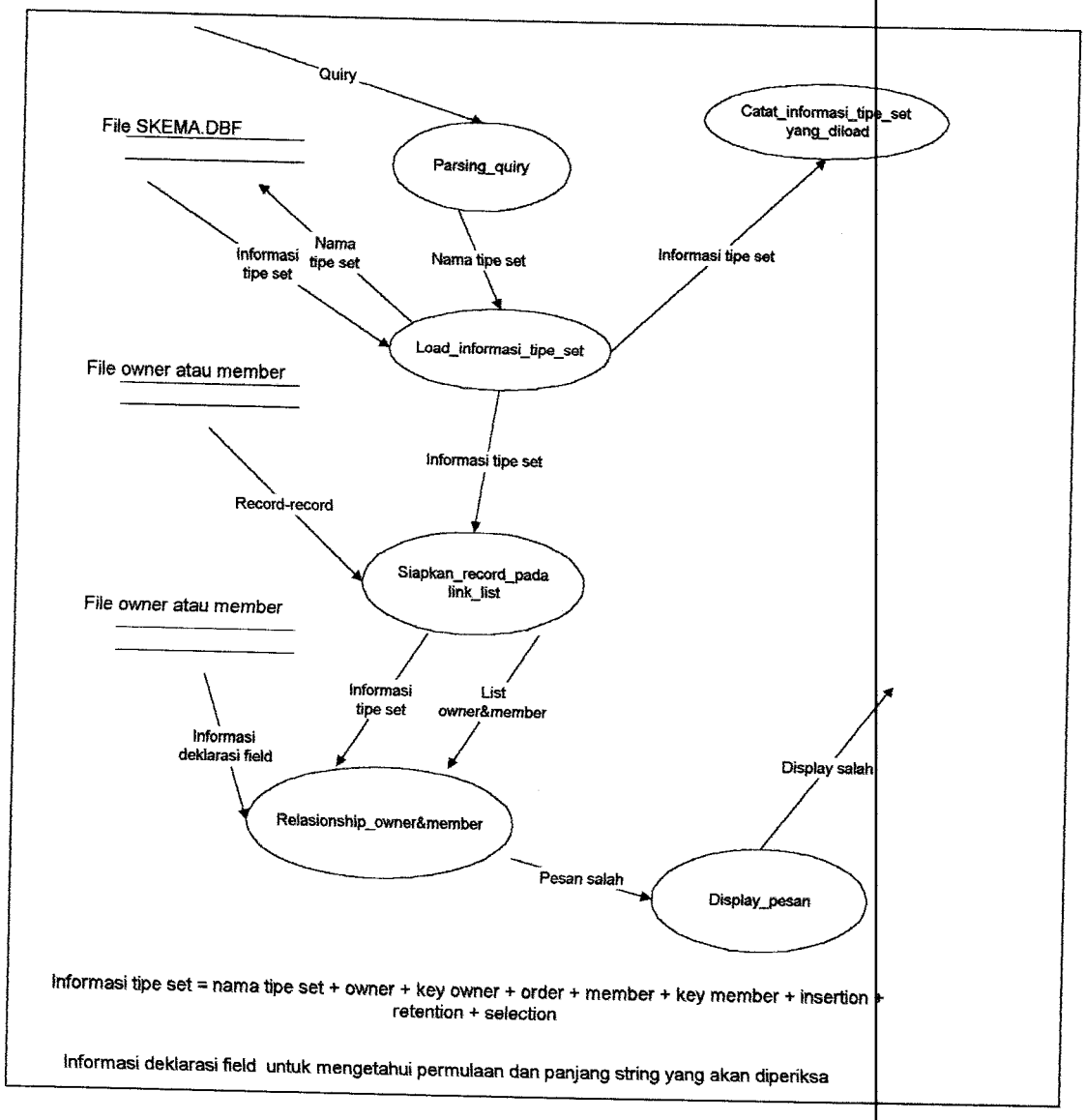


Gambar 3.2 DFD Level 02

Pada diagram aliran data level 02 yang merupakan perincian dari level 01, proses pemasukkan dimulai dengan pembacaan *keyboard* yang ditekan dilanjutkan dengan identifikasi perintah tersebut. Kemudian dilakukan salah satu proses berdasarkan kata kunci dari perintah tersebut. Salah satu proses adalah *build_network, find_any, find_first, find_last, find_next, find_prior, find_prior,*

find_owner, get, store, erase, modify, connect, disconnect, reconnect. Pada bagian berikut akan digambarkan detil setiap proses.

3.1.3. Detil Proses Build_network

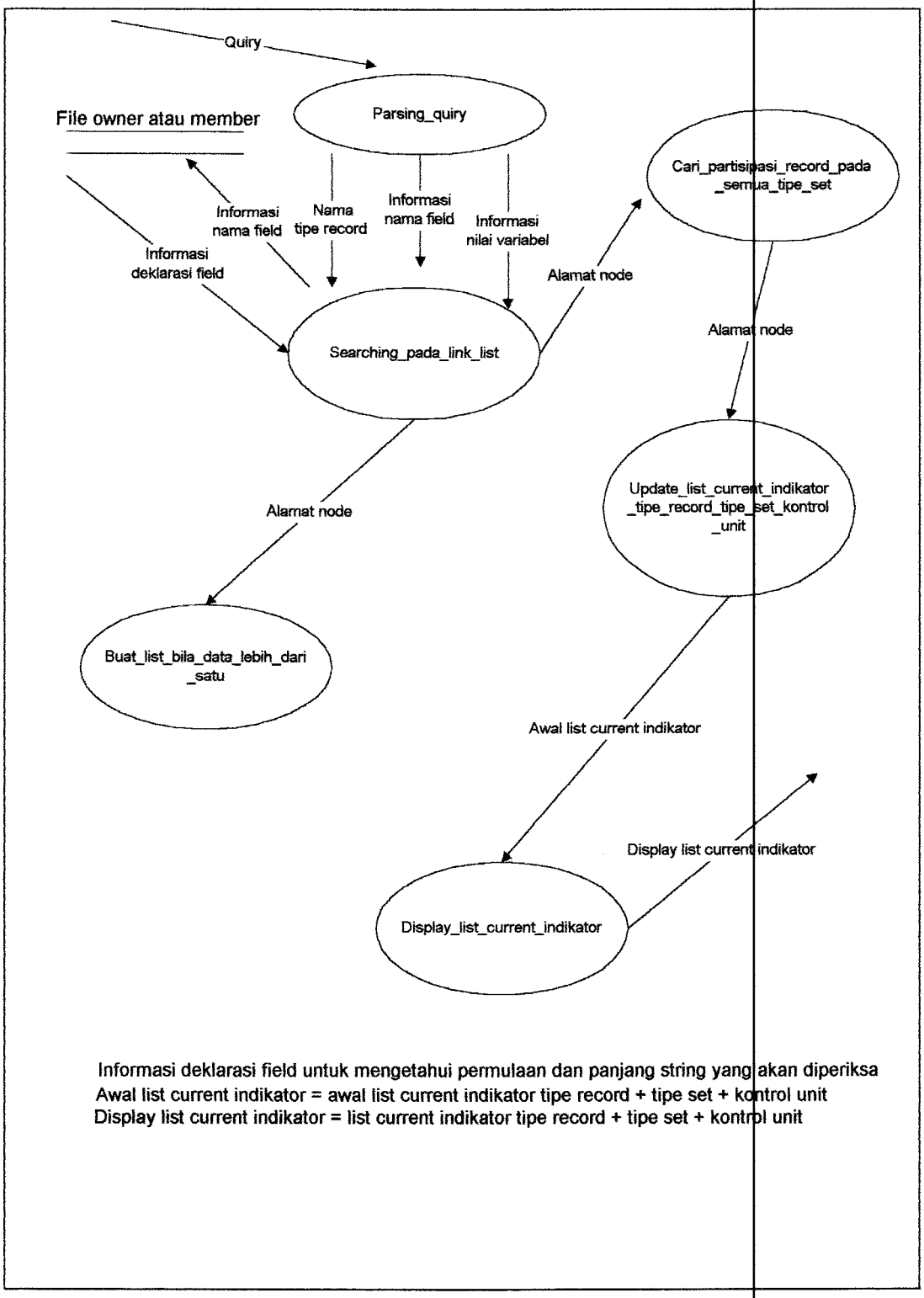


Gambar 3.3 Detil proses build_network

Pada proses *Build_network* ini dimulai dengan prosedur *Parsing_quiry* yang bertujuan menentukan nama tipe set yang akan dibangun struktur *network*nya. Selanjutnya prosedur *Load_ informasi_tipe_set* membaca file SKEMA.DBF untuk

memperoleh informasi tipe set dan menyimpannya melalui prosedur *Catat informasi_tipe_set_yang_diload.* Kemudian prosedur *Siapkan_record_pada_link list* berdasarkan informasi tipe set dilakukan pembacaan record file *owner* serta *member* dan menyimpan record tersebut pada *node link list.* Prosedur *Relasionship_ owner&member* membentuk relasi antara record file *owner* dengan record file *member* berdasarkan informasi tipe set dan informasi deklarasi *field* untuk mengetahui permulaan serta panjang *string* karakter yang akan diseleksi.

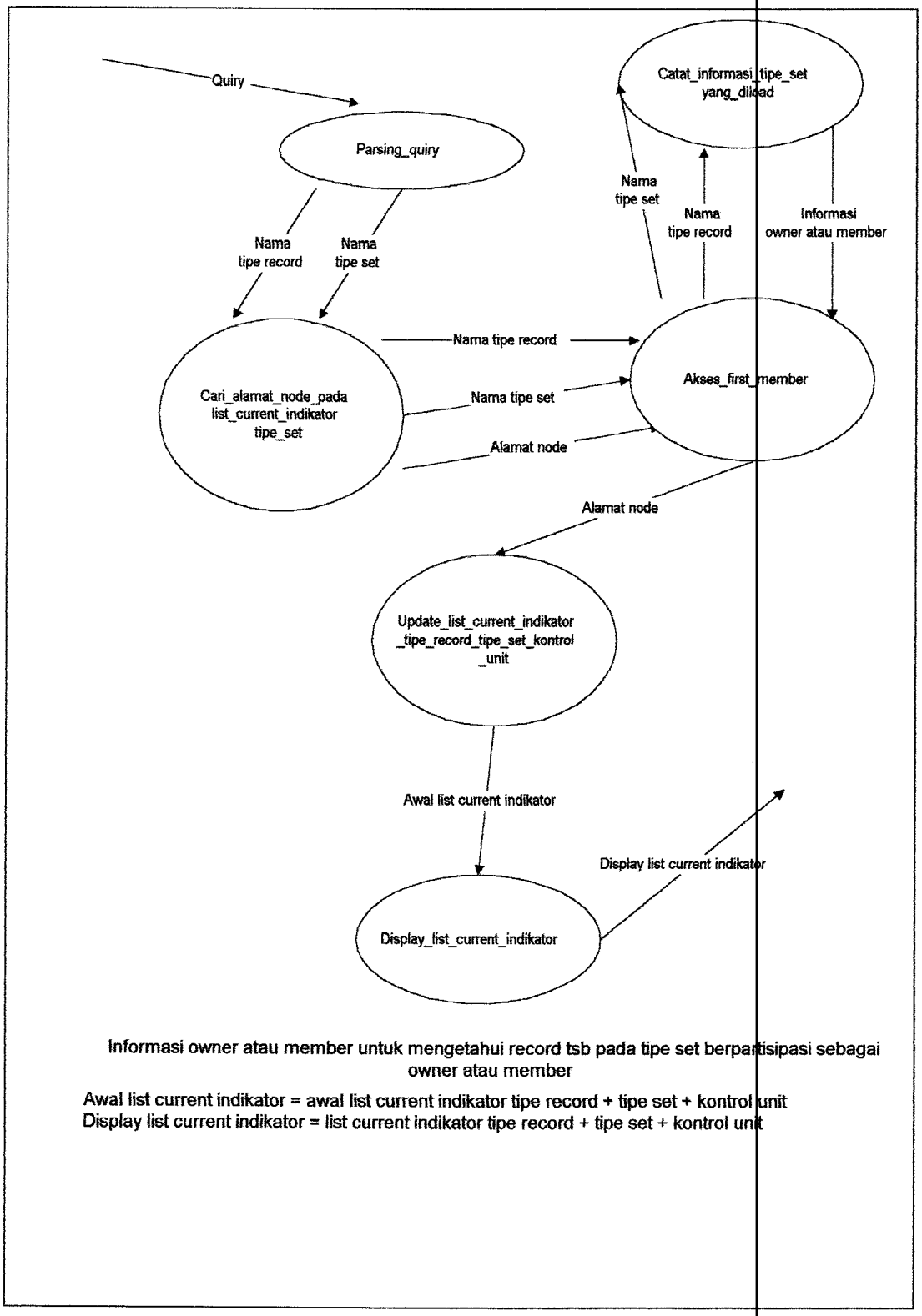
3.1.4 Detil Proses Find any



Gambar 3.4 Detil proses find any

Proses find any diawali prosedur *Parsing _quiry* untuk mendapatkan nama tipe record, informasi nama field dan informasi nilai dari field yang dicari. Kemudian berdasarkan informasi di atas ditambah informasi deklarasi field, prosedur *Searching_ pada_ link_ list* mencari *node* yang memenuhi syarat yang dicari. Setelah ditemukan alamat *node*nya, dilanjutkan prosedur *Cari_ partisipasi record _pada_semua_tipe_set.* Yang dilakukan dalam prosedur ini adalah meneliti partisipasi sebagai *owner* pada indek *owner* record tersebut dan partisipasi sebagai *member* pada indek *member* record tersebut. Dilanjutkan dengan prosedur *Udpate_ list_ indikator _current _tipe _record_ tipe_ set_ kontrol_unit.*

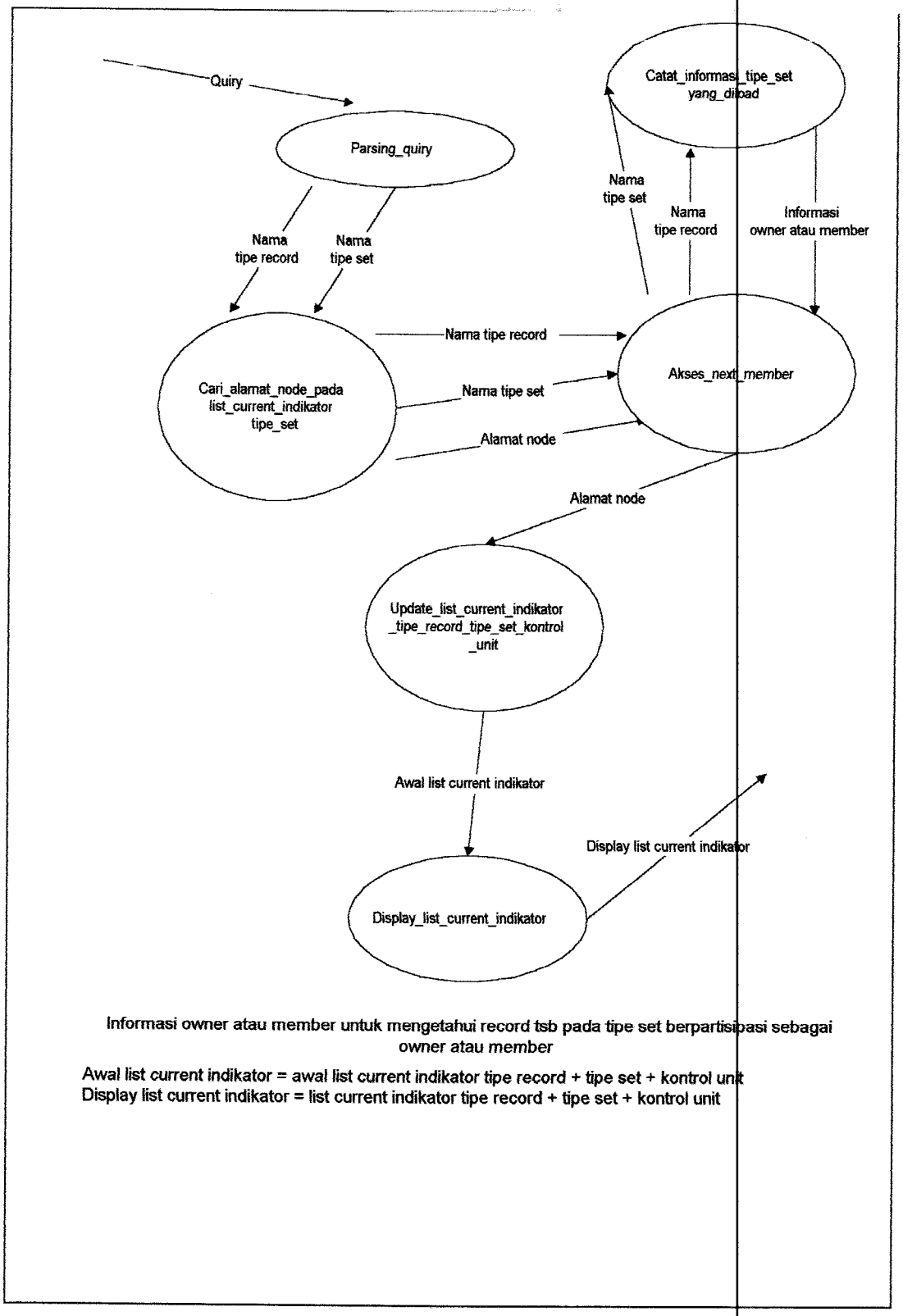
3.1.5 Detil Proses Find First



Gambar 3.5 Detil proses find first

Proses *find first* dimulai dengan prosedur *Parsing_quiry* untuk mengetahui nama tipe record dan nama tipe set. Kemudian dilanjutkan prosedur *Cari_alamat _node_pada_list_current_indikator_tipe_set*, berdasarkan nama tipe set dilakukan pencarian pada list indikator *current* tipe set untuk mengetahui alamat node dari record. Prosedur *Akses_ first_ member* diawali dengan mencari partisipasi nama record sebagai *owner* atau *member* pada prosedur *Catat_informasi_tipe_set_yang_diload.* Bila sebagai *owner* langsung melihat pada indek *member*, tetapi bila sebagai *member* harus dicari *owner*nya dari indek *owner.* Kemudian berdasarkan nama tipe set, nama tipe record, alamat node dicari member pada urutan pertama dari indek member tersebut. Selanjutnya berdasarkan alamat member ke -satu dilakukan prosedur *Update_ list_current indikator_tipe_record_tipe_set_kontrol_unit* . Kemudian ditampilkan list indikator current tipe record, tipe set dan kontrol unit dalam prosedur *Display_list_current_indikator.*

3.1.6 Detil Proses Find Next

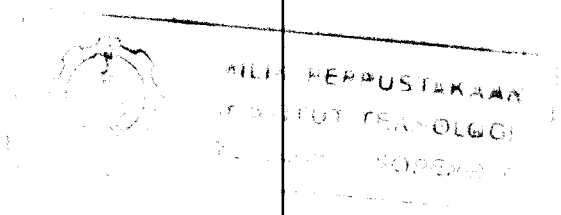


Informasi owner atau member untuk mengetahui record tsb pada tipe set berpartisipasi sebagai owner atau member

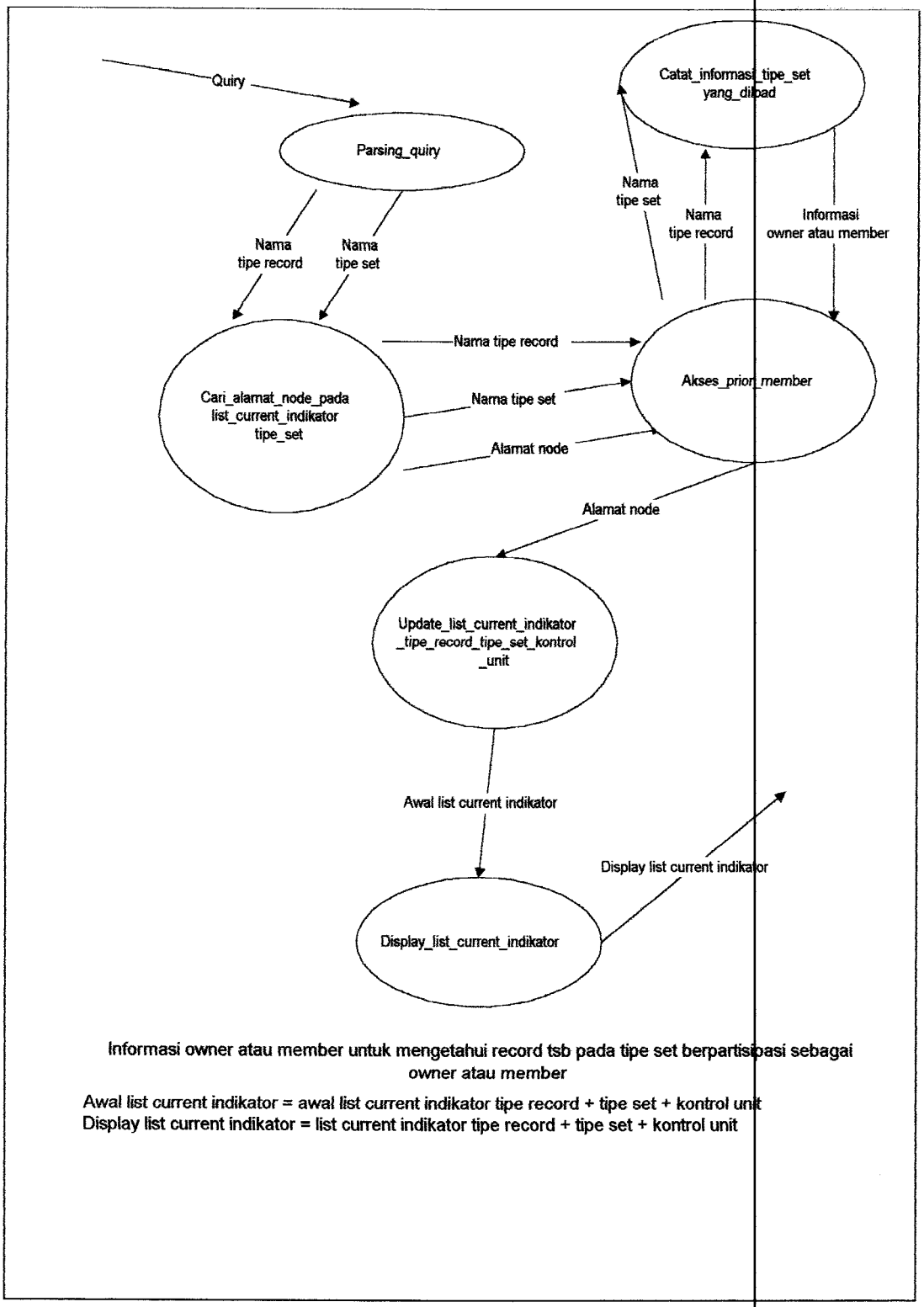
Awal list current indikator = awal list current indikator tipe record + tipe set + kontrol unit
Display list current indikator = list current indikator tipe record + tipe set + kontrol unit

Gambar 3.6 Detil proses find next

Proses *find next* dimulai dengan prosedur *Parsing_quiry* untuk mengetahui nama tipe record dan nama tipe set. Kemudian dilanjutkan prosedur *Cari_alamat _node_pada_list_current_indikator_tipe_set,* berdasarkan nama tipe set dilakukan pencarian pada list indikator *current* tipe set untuk mengetahui alamat node dari record. Prosedur *Akses_ next_ member* diawali dengan mencari partisipasi nama record sebagai *owner* atau *member* pada prosedur *Catat_informasi_tipe_set_yang_diload.* Bila sebagai *owner* langsung melihat pada indek *member*, tetapi bila sebagai *member* harus dicari *owner*nya dari indek *owner.* Kemudian berdasarkan nama tipe set, nama tipe record, alamat node dicari *member* berikutnya setelah *member* pada *current* indikator tipe set dari indek member tersebut. Selanjutnya berdasarkan alamat member yang baru dilakukan prosedur *Update_list_current _indikator_ tipe_ record_ tipe set_kontrol_unit* . Kemudian ditampilkan list indikator current tipe record, tipe set dan kontrol unit dalam prosedur *Display_list_current_indikator.*



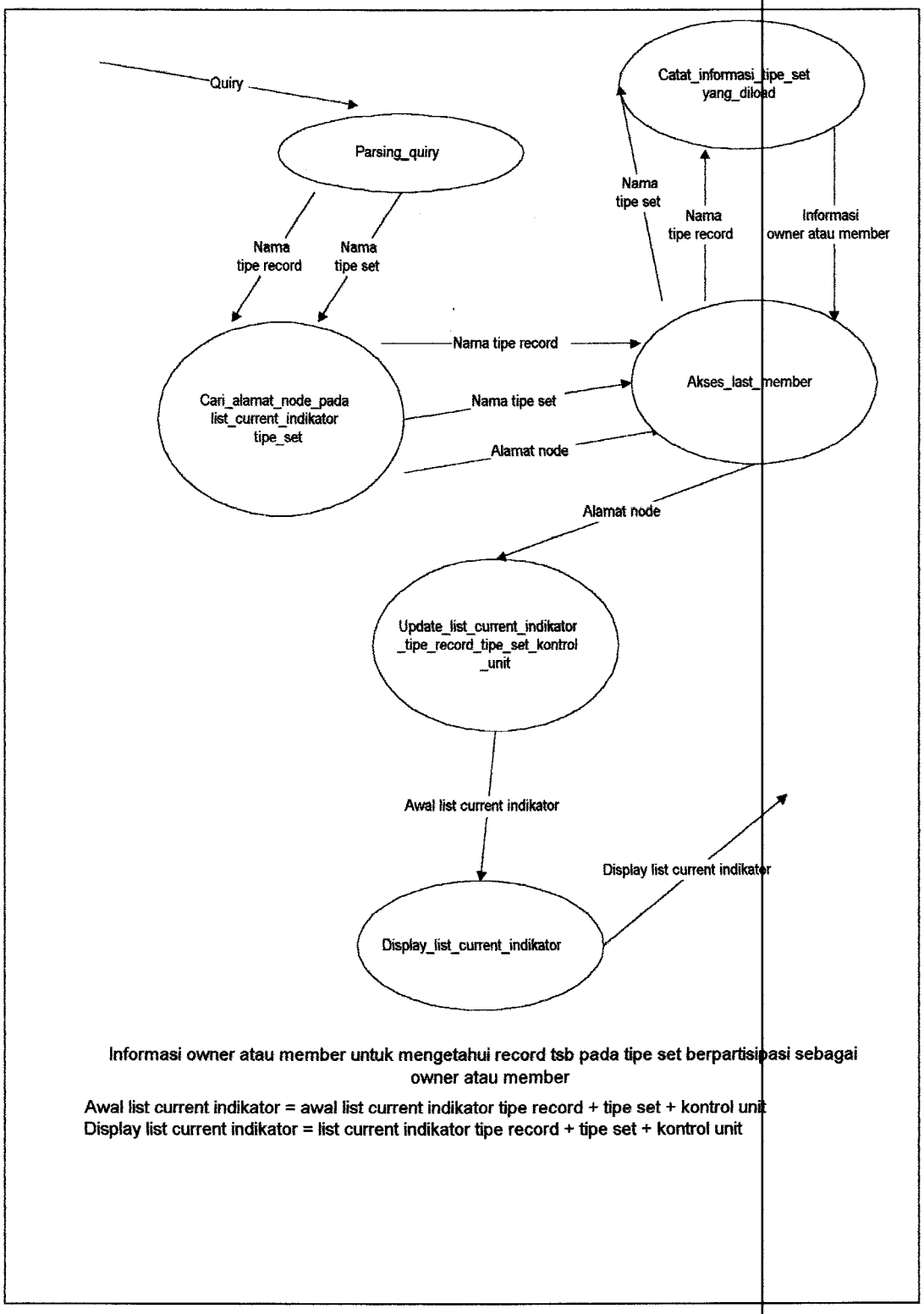
3.1.7 Detil Proses Find Prior



Gambar 3.7 Detil proses find prior

Proses *find prior* dimulai dengan prosedur *Parsing_quiry* untuk mengetahui nama tipe record dan nama tipe set. Kemudian dilanjutkan prosedur *Cari_alamat _node_pada_list_current_indikator_tipe_set*, berdasarkan nama tipe set dilakukan pencarian pada list indikator *current* tipe set untuk mengetahui alamat node dari record. Prosedur *Akses_prior_member* diawali dengan mencari partisipasi nama record sebagai *owner* atau *member* pada prosedur *Catat_informasi_tipe_set_yang_diload.* Bila sebagai *owner* langsung melihat pada indek *member*, tetapi bila sebagai *member* harus dicari *owner*nya dari indek *owner.* Kemudian berdasarkan nama tipe set, nama tipe record, alamat node dicari *member* sebelumnya setelah *member* pada *current* indikator tipe set dari indek member tersebut. Selanjutnya berdasarkan alamat member yang baru dilakukan prosedur *Update_list_current_indikator_tipe_record _tipe_set kontrol unit* . Kemudian ditampilkan list indikator current tipe record, tipe set dan kontrol unit dalam prosedur *Display_list_current_indikator.*

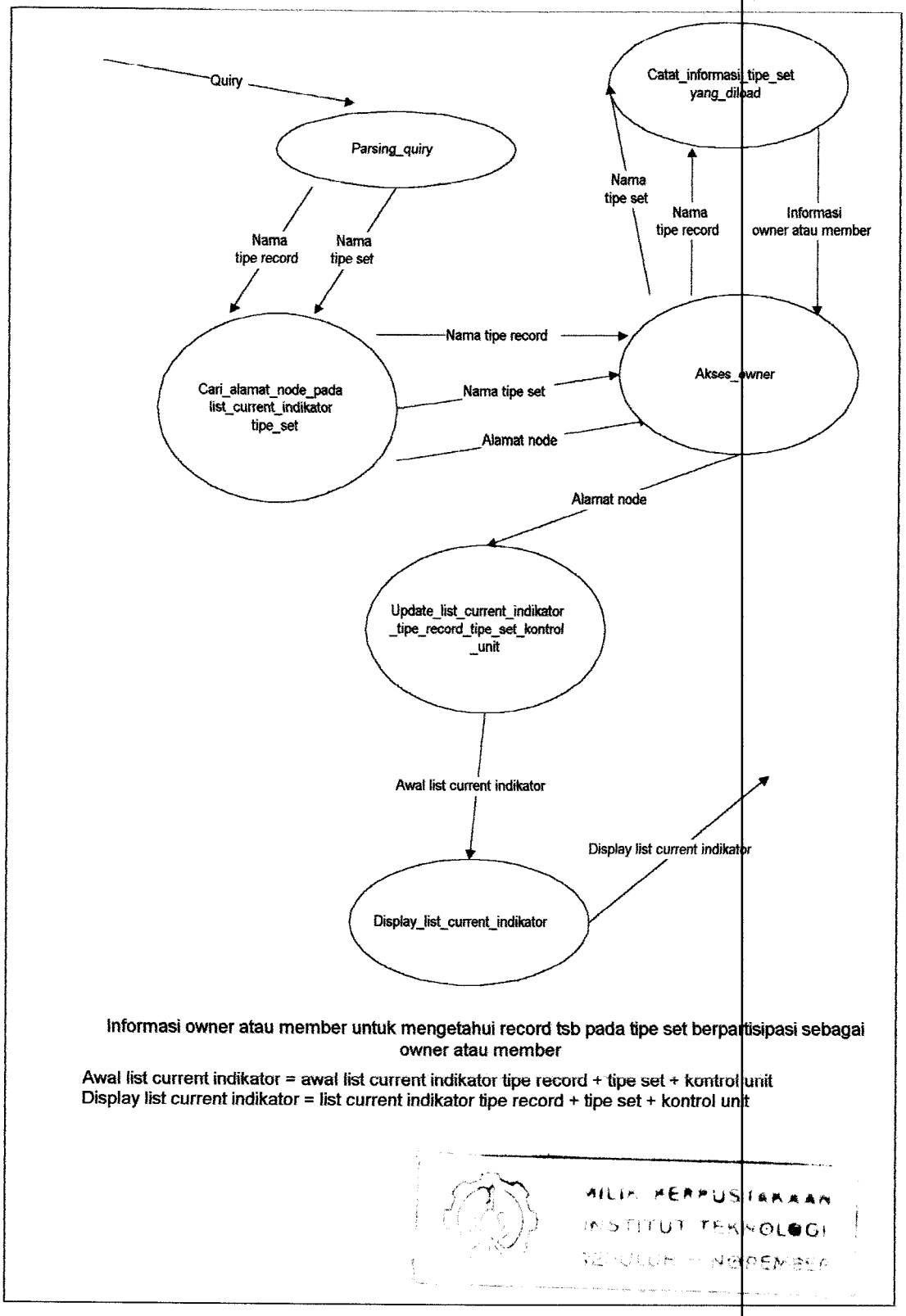
3.1.8 Detil Proses Find Last



Gambar 3.8 Detil proses find last

Proses *find last* dimulai dengan prosedur *Parsing_quiry* untuk mengetahui nama tipe record dan nama tipe set. Kemudian dilanjutkan prosedur *Cari_alamat _node_pada_list_current_indikator_tipe_set*, berdasarkan nama tipe set dilakukan pencarian pada list indikator *current* tipe set untuk mengetahui alamat node dari record. Prosedur *Akses_last_member* diawali dengan mencari partisipasi nama record sebagai *owner* atau *member* pada prosedur *Catat_informasi_tipe_set_yang_diload.* Bila sebagai *owner* langsung melihat pada indek *member*, tetapi bila sebagai *member* harus dicari *owner*nya dari indek *owner.* Kemudian berdasarkan nama tipe set, nama tipe record, alamat node dicari *member* terakhir dari indek member tersebut. Selanjutnya berdasarkan alamat member yang baru dilakukan prosedur *Update_list_current _indikator_tipe record _ tipe_set_control_unit* . Kemudian ditampilkan list indikator current tipe record, tipe set dan kontrol unit dalam prosedur *Display_list_current_indikator.*

3.1.9 Detil Proses Find Owner



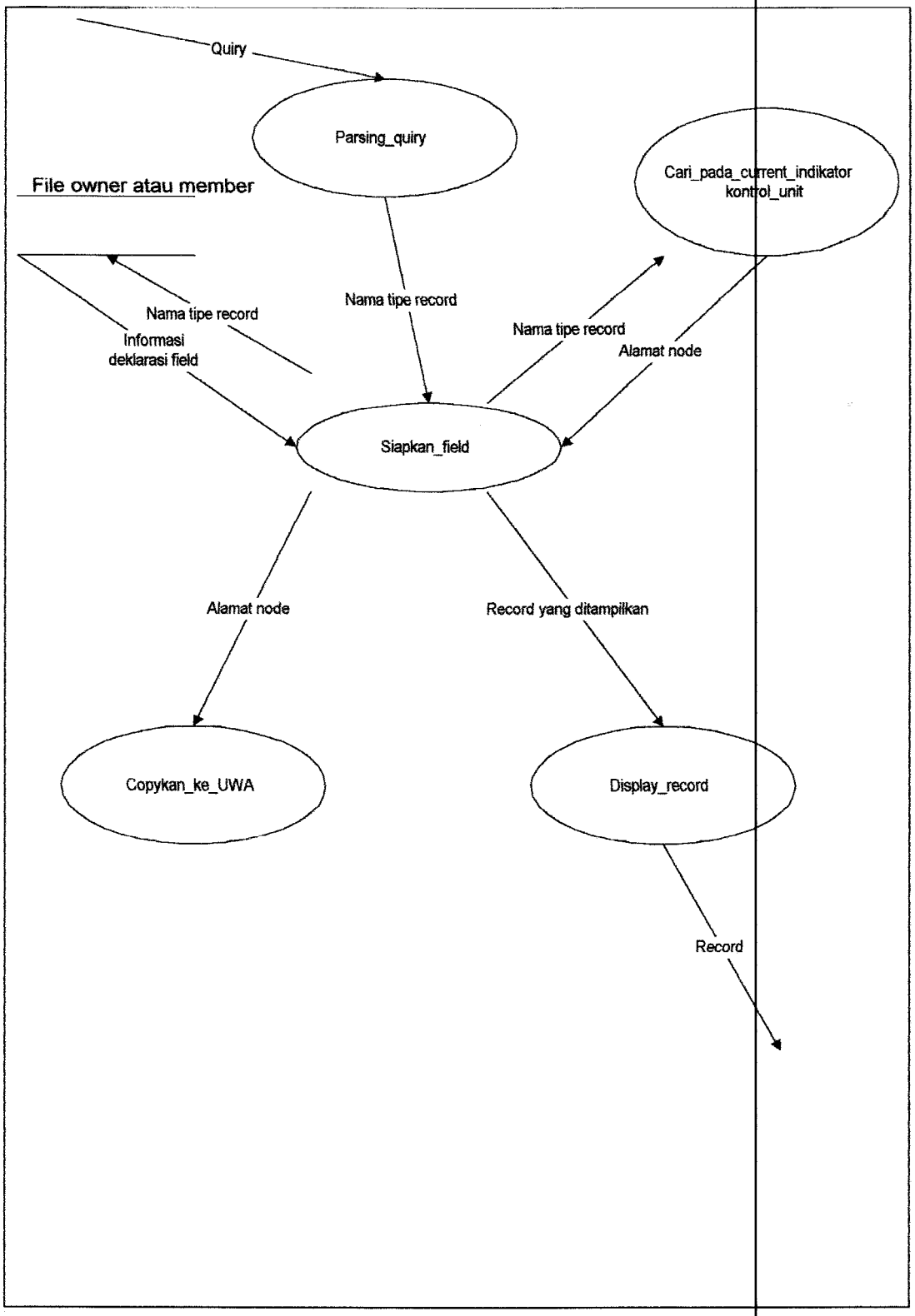
Informasi owner atau member untuk mengetahui record tsb pada tipe set berpartisipasi sebagai owner atau member

Awal list current indikator = awal list current indikator tipe record + tipe set + kontrol unit
Display list current indikator = list current indikator tipe record + tipe set + kontrol unit

Gambar 3.9 Detil proses find owner

Proses *find owner* dimulai dengan prosedur *Parsing_quiry* untuk mengetahui nama tipe record dan nama tipe set. Kemudian dilanjutkan prosedur *Cari_alamat _node_pada_list_current_indikator_tipe_set*, berdasarkan nama tipe set dilakukan pencarian pada list *current* indikator tipe set untuk mengetahui alamat node dari record. Prosedur *Akses_ owner* diawali dengan mencari partisipasi nama record sebagai *owner* atau *member* pada prosedur *Catat_informasi_tipe_set_yang_diload*. Bila sebagai *member* harus dicari *owner*nya dari indek *owner*. Kemudian berdasarkan nama tipe set, nama tipe record, alamat node dicari *owner*. Selanjutnya berdasarkan alamat owner dilakukan prosedur *Update_list_current_indikator tipe_record tipe_set kontrol unit* . Kemudian ditampilkan list indikator current tipe record, tipe set dan kontrol unit dalam prosedur *Display_list_current_indikator*.

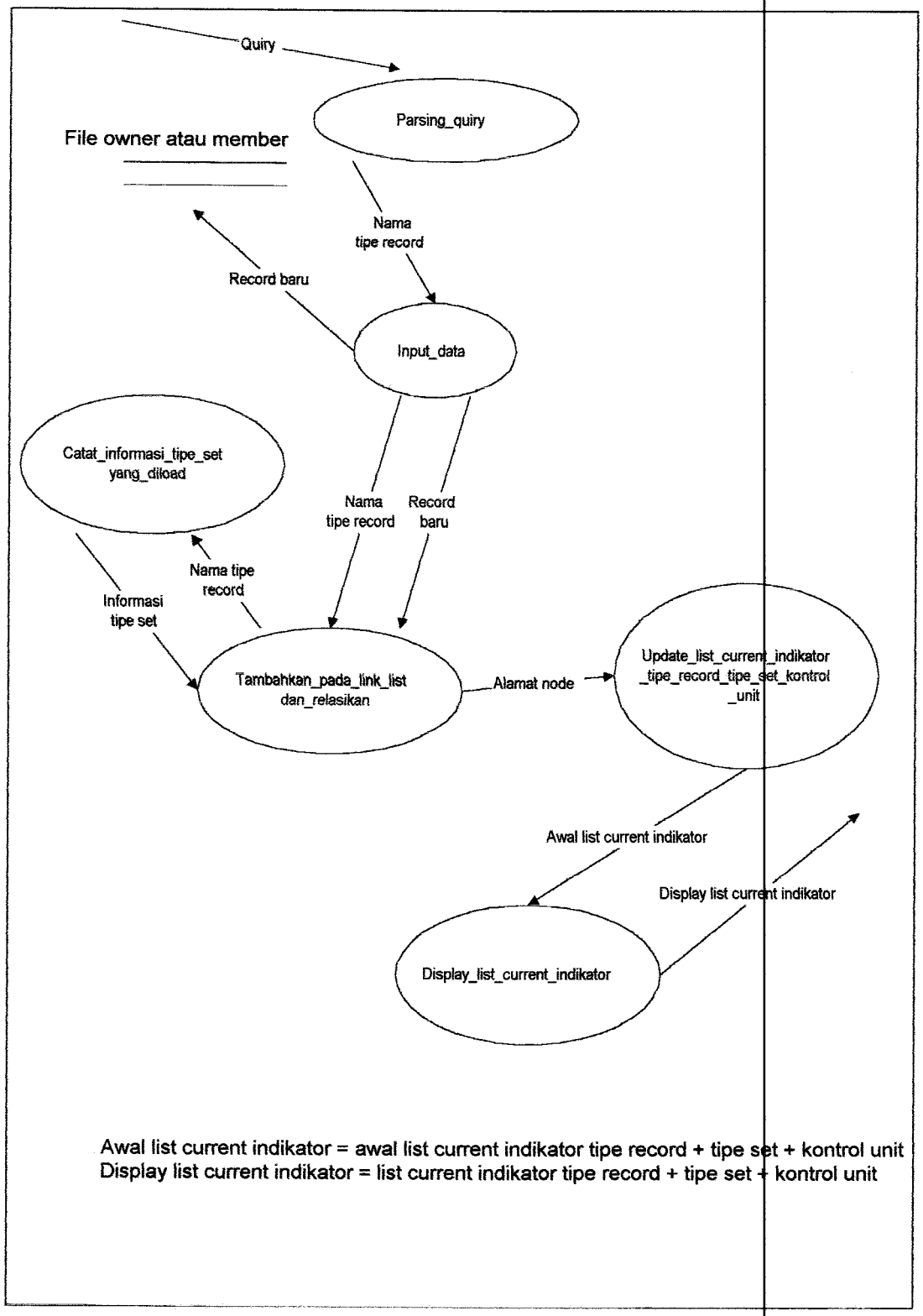
3.1.10. Detil Proses Get



Gambar 3.10 Detil proses get

Proses get diawali dengan prosedur *Parsing_quiry* untuk menentukan nama tipe record. Kemudian dilanjutkan prosedur *Siapkan_field* yang mencetak informasi field dan mencetak nilai tiap field dengan mencari pada *current* indikator kontrol unit. Selanjutnya prosedur *Copykan_ke_UWA* dan prosedur *Display_record.*

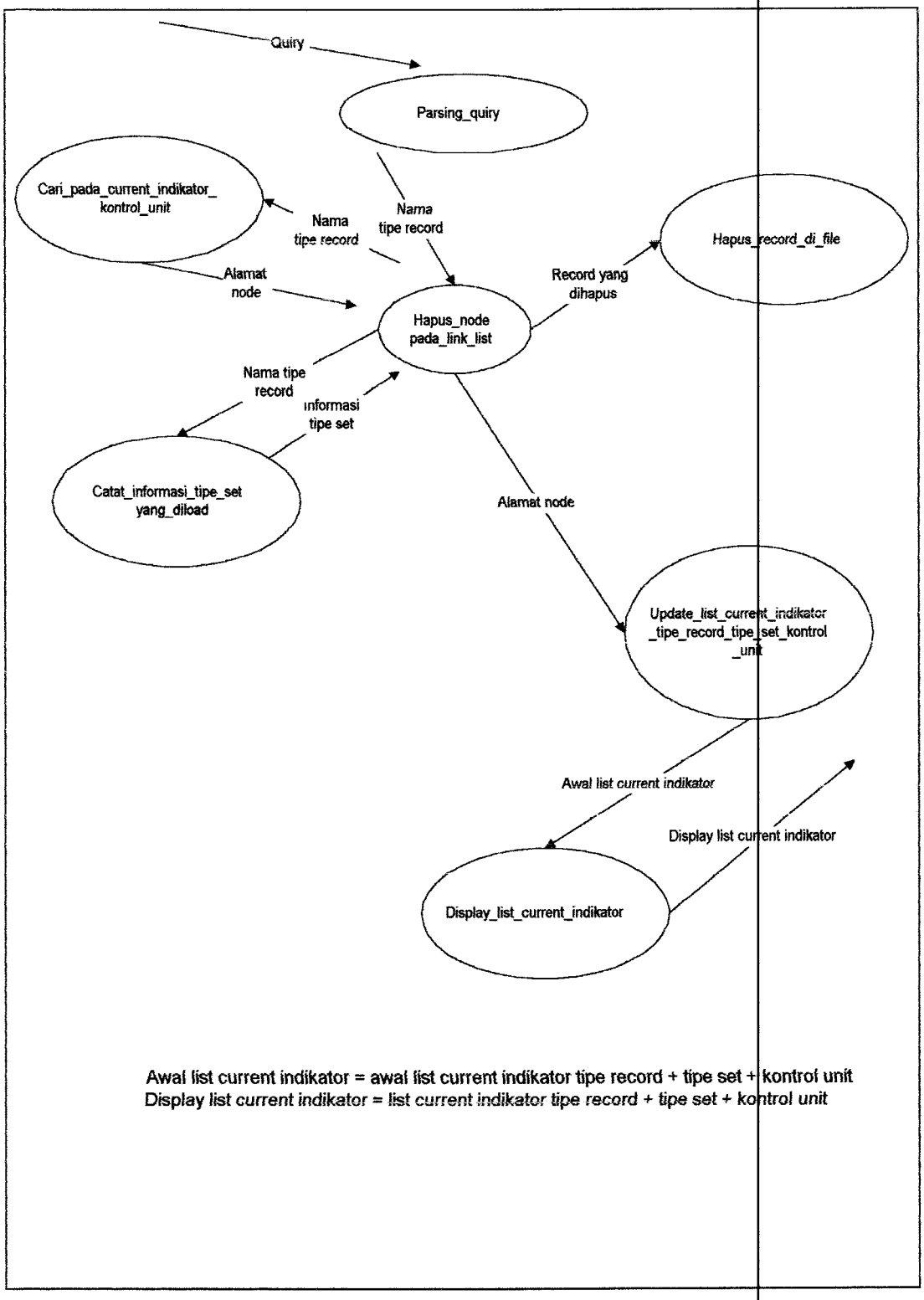
3.1.11. Detil Proses Store



Gambar 3.11 Detil proses store

Proses store diawali dengan prosedur *Parsing_quiry* untuk menentukan nama tipe record yang akan ditambah datanya. Kemudian prosedur *Input_data* yang menyimpan data baru ke dalam file. Selanjutnya prosedur *Tambahkan_pada_list&relasikan* menambahkan record baru pada list dan berdasarkan informasi tipe set direlasikan. Selanjutnya prosedur *Update_list_current_indikator tipe_record tipe_set kontrol _unit* yang meletakkan record baru sebagai current indikator. Prosedur *Display_list_current_indikator.*

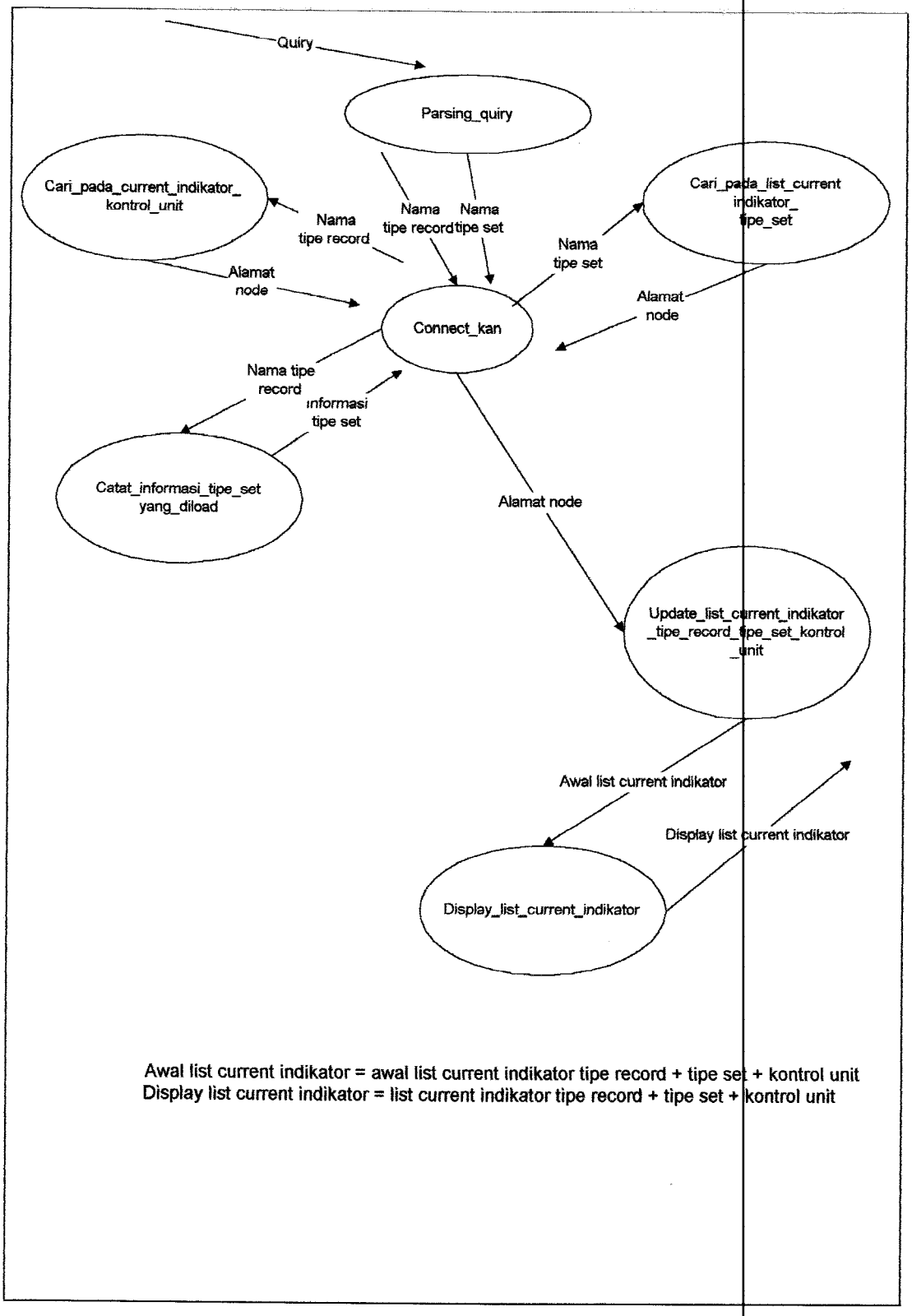
3.1.12 Detil Proses Erase



Gambar 3.12 Detil proses erase

Proses erase diawali prosedur *Parsing_quiry* untuk menentukan nama tipe record. Kemudian prosedur *Hapus_node_pada_list* yang menghapus node pada list tetapi terlebih dahulu harus memeriksa informasi tipe set boleh tidaknya dihapus dari prosedur *Catat_informasi_tipe_set_yang_diload.* Bila berhasil dilanjutkan dengan prosedur *Update_list_current_indikator tipe_record tipe_set _kontrol_unit* dan prosedur *Display_list_current_indikator.*

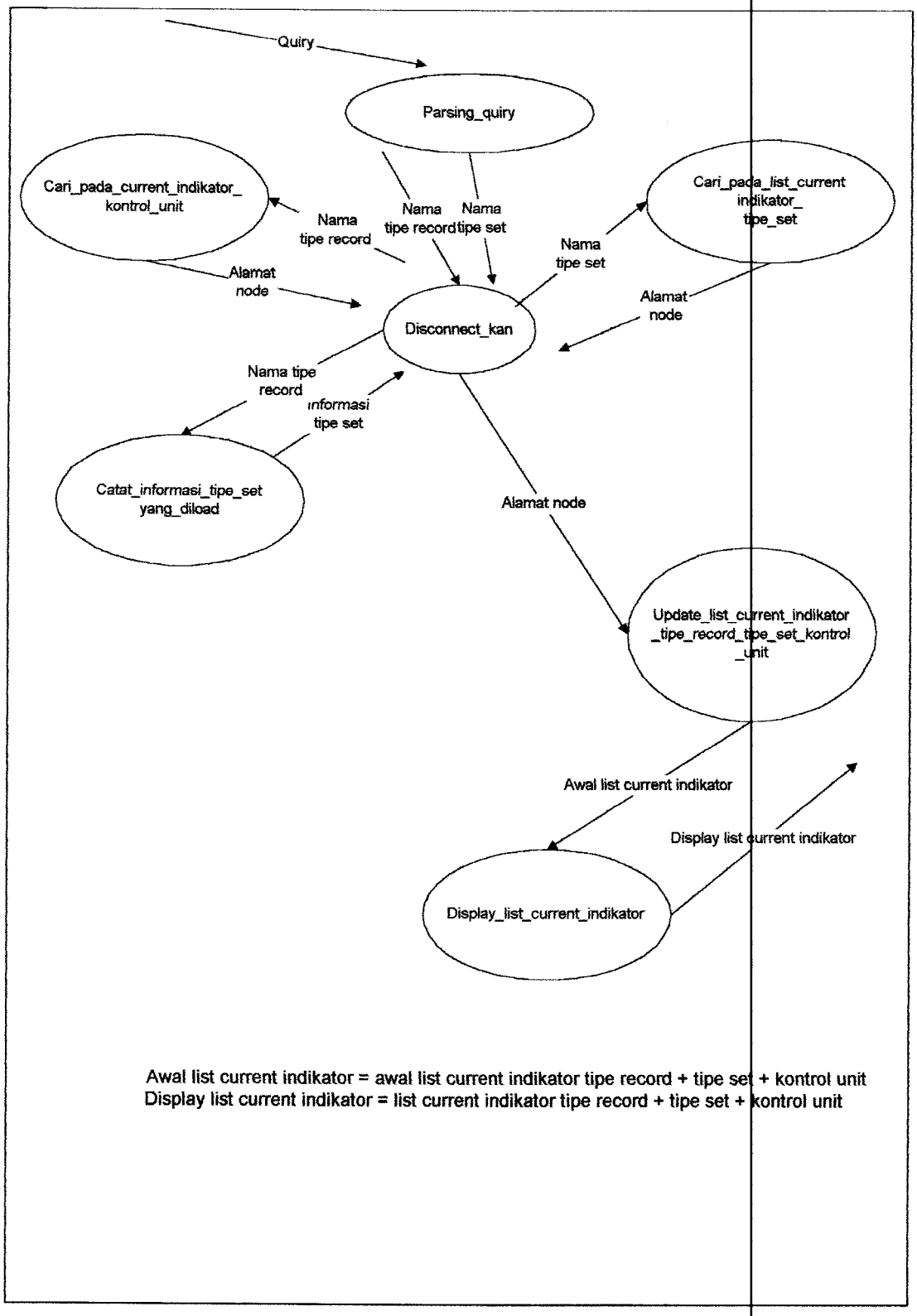
3.1.13 Detil Proses Connect



Gambar 3.13 Detil proses connect

Proses connect diawali dengan *prosedur Parsing_quiry* untuk menentukan nama tipe record yang akan di-connect-kan dan nama tipe set sebagai tujuan. Prosedur *Cari_pada_current_indikator_kontrol_unit* mencari record member yang akan di-connect-kan dan prosedur *Cari_pada_list_current _indikator tipe_set* untuk mencari set occurence sebagai tujuan. Sebelum dilakukan proses connect harus memeriksa informasi tipe set. Selanjutnya prosedur *Update_list_current_indikator tipe_record tipe_set _kontrol_unit* dan prosedur *Display_list_current_indikator.*

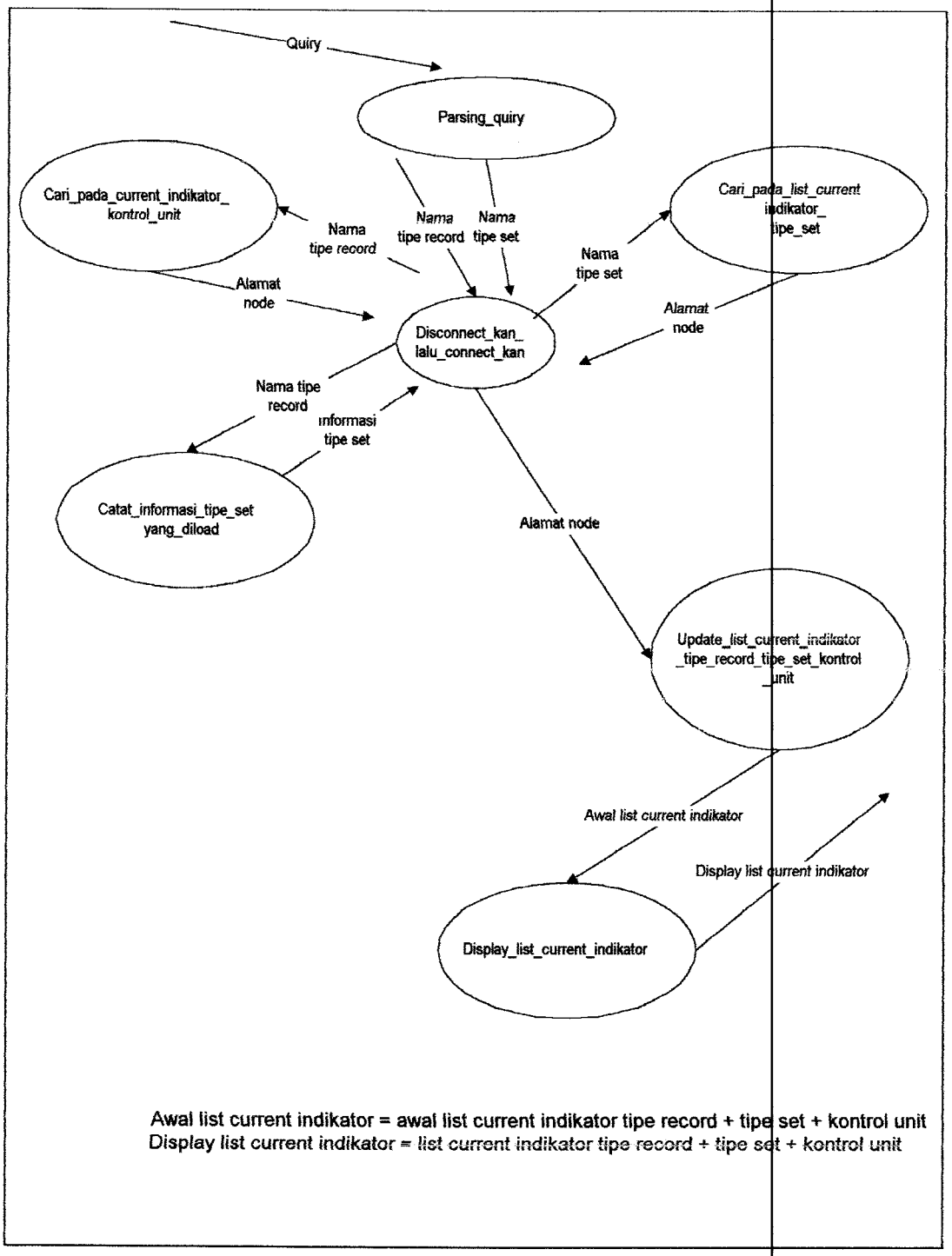
3.1.14 Detil Proses Disconnect



Gambar 3.14 Detil proses disconnect

Proses disconnect diawali dengan prosedur *Parsing_quiry* untuk menentukan nama tipe record yang akan di-connect-kan dan nama tipe set sebagai tujuan. Prosedur *Cari_pada_current_indikator _kontrol_unit* mencari record member yang akan di-connect-kan dan prosedur *Cari_pada_list_current_indikator tipe_set* untuk mencari set occurence . Sebelum dilakukan proses disconnect harus memeriksa informasi tipe set. Selanjutnya prosedur *Update_list_current_indikator tipe_record tipe_set _kontrol_unit* dan prosedur *Display_list_current_indikator.*

3.1.15 Detil Proses Reconnect



Gambar 3.15 Detil proses reconnect

Proses reconnect merupakan proses berurutan dari proses disconnect dan connect.

3.2 Diskripsi Fungsi

3.2.1 Diskripsi fungsi pada proses Build_network

Proses Build_network adalah proses menyusun record-record dari file *owner* dan file *member* membentuk relasi seperti pada implementasi fisik model data *network* dengan indek. Prosedur utama dalam proses *build network* adalah :

-prosedur *Load_informasi_tipe_set*(nama tipe set) yaitu prosedur mencari deklarasi tipe set dalam file SKEMA.DBF

-prosedur *Catat_informasi_tipe_set_yang diload*(informasi tipe set) yaitu prosedur mencatat deklarasi tipe set dalam array SET.

-prosedur *Siapkan_record_pada_link_list*(Informasi tipe set) yaitu prosedur meletakkan item data record dari file *owner* dan file *member* pada *node link list.*

-prosedur *Relasionship_owner&member*(Informasi tipe set, list owner. list member) yaitu prosedur membentuk relasi antara record file *owner* dan file *member* berdasarkan informasi tipe set seleksi dan *order* set.

Algoritma setiap prosedur diatas adalah:

```
prosedur Load_informasi_tipe_set(nama_tipe_set)
begin
    Open file skema;
    do
        if(ditemukan nama_set)
        Catat_informasi_tipe_set_yang_diload(informasi tipe set);
    while(!EOF skema);
end prosedur
```

```
prosedur Catat_informasi_tipe_set_yang_diload(informasi tipe set)
begin
    set[indek].nama_set=informasi tipe set->nama_set;
    set[indek].owner=informasi tipe set->owner;
    set[indek].key_owner=informasi tipe set->key_owner;
    set[indek].order=informasi tipe set->order;
    set[indek].member=informasi tipe set->member;
    set[indek].key_member=informasi tipe set->key_member;
    set[indek].insertion=informasi tipe set->insertion;
    set[indek].retention=informasi tipe set->retention;
    set[indek].selection=informasi tipe set->selection;
end prosedur

prosedur Siapkan_record_pada_link_list(informasi set)
begin
chek apakah sudah ada list berisi record dari file info_set->owner;
    if(tidak_ada)
    begin
        Open file(informasi tipe set->owner);
        do
        record diletakkan pada node list;
        while(!EOF file(informasi tipe set->owner);
    end
    chek apakah sudah ada list berisi record dari file informasi tipe set->member;
    if(tidak_ada)
    begin
        Open file(informasi tipe set->member);
        do
            record diletakkan pada node list;
        while(!EOF file(informasi tipe set->member);
    end
end prosedur


prosedur Relasionship_owner&member(informasi tipe set, list owner, list member)
begin
    parsing informasi tipe set->selection untuk mengetahui field owner dan member
    sebagai syarat relasi;
    Open file(informasi tipe set->owner &informasi tipe set->member) untuk
    mengetahui permulaan dan panjang string untuk matching dalam node list; /*
    dari info field.name,field.type,field.leng,field.dec*/
    letakkan pointer1 pada node ke-1 list_owner;
    while(!End list_owner)
    begin
        letakkan pointer2 pada node ke-1 list_member;
        while(!End list_member)
```

```
        begin
            if(pointer1->data==pointer2->data)
            begin
                info_set->order untuk mengetahui aturan penyisipan member;
                update pointer1->indek_member;
                update pointer2->indek_owner;
            end if
            letakkan pointer2 ke node next list_member;
        end loop list_member
        letakkan pointer1 ke node next list_owner;
    end loop list_owner
end prosedur
```

3.2.2 Diskripsi fungsi pada proses Find_any

Proses Find_any yaitu proses mencari *node* pada *link list* yang memenuhi syarat yang dicari. Prosedur utama dalam proses find any ini terdiri :

-prosedur *Searching_di_link_list*(nama tipe record,informasi nama field,informasi nilai variabel) yaitu prosedur yang mencari *node* di *link list* yang memenuhi syarat yang dicari. Prosedur ini diawali dengan mencari informasi permulaan dan panjang string yang dicari dari informasi deklarasi *field* file tersebut.

-prosedur *Cari_partisipasi_record_pada_semua_tipe_set*(alamat node) yaitu prosedur yang mencari partisipasi record tersebut pada tipe set dengan melihat pada indek *owner* dan indek *member node* tersebut.

-prosedur

Update_list_current_indikator_tipe_record_tipe_set_kontrol_unit(alamat node) yaitu prosedur memperbaharui *list current* indikator.

Algoritma setiap prosedur diatas adalah:

```
prosedur Searching_pada_link_list(nama tipe record,informasi nama field,
informasi nilai variabel)
begin
    Open file(nama tipe record) untuk mengetahui permulaan dan panjang string
    yang akan dilakukan searching pada node_list;
    letakkan pointer pada node ke-1 list;
    while(!end node list)
    begin
        if(ditemukan node list dimana informasi nilai variabel dipenuhi)
            Buat_list_bila_data_lebih_dari_1(alamat node);
        if(pertama kali ditemukan)
            Cari_partisipasi_record_pada_semua_tipe_set(alamat node);
        letakkan pointer pada node next list;
    end while
end prosedur

prosedur Cari_partipasi_record_pada_semua_tipe_set(alamat node)
begin
    Update_list _current_indikator_tipe_record_tipe_set_kontrol_unit (alamat
    node);
    letakkan pointer pada indek_owner ;
    while(!end indek_owner)
    begin
        Update_list _current_indikator_tipe_record_tipe_set_kontrol_unit (alamat
        node);
        letakkan pointer ke node next indek_owner;
    end while
    letakkan pointer pada indek_member;
    while(!end indek_member)
    begin
        Update_list _current_indikator_tipe_record_tipe_set_kontrol_unit (alamat
        node);
        letakkan pointer ke node next indek_member;
    end while
end prosedur
```

3.2.3 Diskripsi fungsi pada proses Find_first

Proses Find_first adalah proses mencari record *member* langsung setelah record *owner* pada sebuah *set occurence* yang ada di *current indikator* tipe set. Prosedur utama dalam proses find first adalah :

-prosedur *Cari_alamat_node_pada_list_current_indikator_tipe_set*(nama tipe record,nama tipe set) yaitu prosedur mencari alamat *node* pada *current* tipe set sesuai nama tipe set yang dicari.

-prosedur *Akses_first_member*(alamat_node,nama tipe record,nama tipe set) yaitu prosedur yang dimulai dengan meneliti apakah *node* berpartisipasi sebagai *owner* atau *member*. Bila sebagai o*wner, first member* langsung dicari pada indek *member*nya. Bila sebagai *member* harus dicari *owner*nya pada indek *owner*, kemudian *first member* dicari pada indek *member* dari *owner*nya tersebut.

Algoritma setiap prosedur diatas adalah:

```
prosedur Cari_alamat_node_pada_list_current_indikator_tipe_set(nama tipe
record,nama tipe set)
begin
    cari record pada nama_set yang terakhir diakses dalam
    list_current_indikator_tipe_set;
    Akses_first_member(alamat_node,nama tipe record,nama tipe set);
end prosedur

prosedur Akses_first_member(alamat node,nama tipe record,nama tipe set)
begin
    tentukan apakah record yang diperoleh owner atau member dengan mengecek
    pada Catat_informasi_tipe_set_yang_diload;
    if(owner)
        periksa node ke-1 pada indek_member dimana nama_setnya sesuai;
    if(member)
    begin
```

cari ownernya dengan memeriksa indek_owner yang sesuai nama_set;
lalu periksa node ke-1 indek_member pada ownernya;
end
Update_list _current_indikator_tipe_record_tipe_set_kontrol_unit (alamat node);
end prosedur

3.2.4 Diskripsi fungsi pada proses Find_last

Proses Find_last adalah proses mencari record *member* terakhir pada sebuah *set occurence* yang ada di *current* tipe set. Prosedur utama dalam proses find last adalah :

-prosedur *Cari_alamat_node_pada_list_current_indikator_tipe_set*(nama tipe record,nama tipe set) yaitu prosedur mencari alamat *node* pada *current* tipe set sesuai nama set yang dicari.

-prosedur *Akses_last_member*(alamat node,nama tipe record,nama tipe set) yaitu prosedur yang dimulai dengan meneliti apakah *node* berpartisipasi sebagai *owner* atau *member*. Bila sebagai *owner*, *last member* langsung dicari pada indek *member*nya. Bila sebagai *member* harus dicari *owner*nya pada indek *owner*, kemudian *last member* dicari pada indek *member* dari *owner*nya tersebut.

Algoritma setiap prosedur diatas adalah:

prosedur Cari_alamat_node_pada_list_current_indikator_tipe_set(nama tipe record,nama tipe set)
begin
cari record pada nama_set yang terakhir diakses dalam list current_indiktor_tipe_set;
Akses_last_member(alamat_node,nama tipe record,nama tipe set);
end prosedur

prosedur Akses_last_member(alamat node,nama tipe record,nama tipe set)
begin
tentukan apakah record yang diperoleh owner atau member dengan mengecek pada Catat_informasi_tipe_set_yang_diload;

if(owner)
periksa node last pada indek_member dimana nama_setnya sesuai;
if(member)
begin
cari ownernya dengan memeriksa indek_owner yang sesuai nama_set;
lalu periksa node last indek_member pada ownernya;
end
Update_list _current_indikator_tipe_record_tipe_set_kontrol_unit (alamat node);
end prosedur

3.2.5 Diskripsi fungsi pada proses Find_next

Proses Find_next adalah proses mencari record *member* setelah record pada sebuah *set occurence* yang ada di *current* tipe set. Prosedur utama dalam proses find next adalah :

-prosedur *Cari_alamat_node_pada_list_current_indikator_ tipe_set*(nama tipe record,nama tipe set) yaitu prosedur mencari alamat *node* pada *current* tipe set sesuai nama set yang dicari.

-prosedur *Akses_next_member*(alamat node,nama tipe record,nama tipe set) yaitu prosedur yang dimulai dengan meneliti apakah *node* berpartisipasi sebagai *owner* atau *member*. Bila sebagai *owner, next member* langsung dicari pada indek *member*nya. Bila sebagai *member* harus dicari *owner*nya pada indek *owner*, kemudian *next member* dicari pada indek *member* dari *owner*nya tersebut.

Algoritma setiap prosedur diatas adalah:

prosedur Cari_alamat_node_pada_list_current_indikator_tipe_set(nama tipe record,nama tipe set)
begin
cari record pada nama_set yang terakhir diakses dalam list Set_current_set;
Akses_next_member(alamat node,nama tipe record,nama tipe set);
end prosedur

```
prosedur Akses_next_member(alamat node,nama tipe record,nama tipe set)
begin
    tentukan apakah record yang diperoleh owner atau member dengan mengecek
    pada Catat_informasi_tipe_set_yang_diload;
    if(owner)
        periksa node ke-1 pada indek_member dimana nama_setnya sesuai;
    if(member)
    begin
        cari ownernya dengan memeriksa indek_owner yang sesuai nama_set;
        lalu periksa node next setelah node yang berisi pointer record yang
        diperoleh indek_member pada ownernya;
    end
    Update_list _current_indikator_tipe_record_tipe_set_kontrol_unit (alamat
    node);
end prosedur
```

3.2.6 Diskripsi fungsi pada proses Find_prior

Proses Find_prior adalah proses mencari record *member* sebelum record pada sebuah *set occurence* yang ada di *current* tipe set. Proses utama dalam proses find prior adalah :

-prosedur *Cari_alamat_node_pada_list_current_indikator_tipe_set*(nama tipe record,nama tipe set) yaitu prosedur mencari alamat *node* pada *current* tipe set sesuai nama set yang dicari.

-prosedur *Akses_prior_member*(alamat node,nama tipe record,nama tipe set) yaitu prosedur yang dimulai dengan meneliti apakah node berpartisipasi sebagai *owner* atau *member*. Bila sebagai *owner*, *prior member* langsung dicari pada indek *member*nya. Bila sebagai *member* harus dicari *owner*nya

pada indek *owner*, kemudian *prior member* dicari pada indek *member* dari *owner*nya tersebut.

Algoritma setiap prosedur diatas adalah:

```
prosedur Cari_alamat_node_pada_list_current_indikator_tipe_set(nama tipe record,nama tipe set)
begin
    cari record pada nama_set yang terakhir diakses dalam list current_indikator_tipe_set;
    Akses_prior_member(alamat node,nama tipe record,nama tipe set);
end prosedur

prosedur Akses_prior_member(alamat node,nama tipe record,nama tipe set)
begin
    tentukan apakah record yang diperoleh owner atau member dengan mengecek pada Catat_informasi_tipe_set_yang_diload;
    if(owner)
        periksa node last pada indek_member dimana nama_setnya sesuai;
    if(member)
    begin
        cari ownernya dengan memeriksa indek_owner yang sesuai nama_set;
        lalu periksa node prior dari node yang berisi addres record yang diperoleh
        indek_member pada ownernya;
    end
    Update_list _current_indikator_tipe_record_tipe_set_kontrol_unit (alamat node);
end prosedur
```

3.2.7 Diskripsi fungsi pada proses Find_owner

Proses Find_owner adalah proses mencari record *owner* pada sebuah *set occurence* yang ada di *current* tipe set.

Prosedur utama dalam proses find owner adalah :

-prosedur *Cari_alamat_node_pada _list_indikator_current_tipe_set*(nama tipe record,nama tipe set) yaitu prosedur mencari alamat *node* pada *current* tipe set sesuai nama set yang dicari.

-prosedur *Akses_owner*(alamat node,nama tipe record,nama tipe set) yaitu prosedur yang dimulai dengan meneliti apakah *node* berpartisipasi sebagai *owner* atau *member*. Bila sebagai *member* dicari pada indek *owner* sesuai nama set yang dicari.

Algoritma setiap prosedur diatas adalah:

```
prosedur Cari_alamat_node_pada_list_current_indikator_tipe_set(nama tipe
record,nama tipe set)
begin
    cari record pada nama_set yang terakhir diakses dalam list Set_current_set;
    Akses_owner( alamat node,nama tipe record,nama tipe set);
end prosedur

prosedur Akses_owner(alamat node,nama tipe record,nama tipe set)
begin
    tentukan apakah record yang diperoleh owner atau member dengan mengecek
    pada Catat_informasi_tipe_set_yang_diload;
    if(member)
        cari ownernya dengan memeriksa indek_owner yang sesuai nama_set;
    Update_list _current_indikator_tipe_record_tipe_set_kontrol_unit (alamat
    node);
end prosedur
```


3.2.8 Diskripsi fungsi proses Get

Proses Get adalah proses memindahkan record pada kontrol unit ke dalam unit *work area* sehingga dapat dimanipulasi.

Prosedur utama dalam proses get adalah :

-prosedur *Siapkan_field* (nama tipe record) yaitu prosedur menampilkan nama *field* dari record tersebut. Selanjutnya menampilkan nilai *field*nya. Record ini

adalah record yang terdapat pada kontrol unit. Untuk itu dilakukan prosedur *Cari_pada_current_indikator_kontrol_unit.*

-prosedur *Copykan_ke_UWA* yaitu prosedur menyimpan record pada kontrol unit ke *user work area(UWA)* sehingga record tersebut dapat dimanipulasi.

Algoritma proses get adalah :

```
prosedur Siapkan_field(nama tipe record)
begin
    Open file(nama tipe  record);
    tampilkan nama_field;
    cari addres_node dimana record yang terakhir diakses pada control_unit;
    tampilkan nilai setiap field dengan memilah addres_node->data berdasarkan
    permulaan dan panjang field.
end prosedur
```

3.2.9 Diskripsi fungsi pada proses Store

Proses Store adalah proses menyimpan sebuah record baru dan merelasikan record tersebut dengan record lain berdasarkan deklarasi tipe set.

Prosedur utama yang terdapat pada proses Store adalah :

-prosedur Input_data(nama_record) yaitu prosedur memasukkan data berdasarkan item data dalam record.

-prosedur Tambahkan_pada_list_dan_relasikan(nama tipe record,data baru) yaitu prosedur menyimpan data record pada *node link list* dan membentuk relasi berdasarkan *set order,set* seleksi.

Algoritma setiap prosedur diatas adalah:

```
prosedur Input_data(nama tipe record)
begin
    Open file(nama tipe record);
    tampilkan field-fieldnya;
    inputkan data setiap field;
    simpan pada file;
    Tambahkan_pada_list_dan_relasikan(nama tipe record,data_baru);
    Update_list    _current_indikator_tipe_record_tipe_set_kontrol_unit    (alamat
    node);
end prosedur
```

```
prosedur Tambahkan_pada_list_dan_relasikan(nama tipe record,data_baru)
begin
    tambahkan data_baru pada node terakhir dari list tersebut;
    periksa Catat_informasi_tipe_set_yang_diload;
    tentukan apakah record baru owner atau member;
    periksa informasi tipe set->insertion;
    if(automatic)
    begin
        periksa informasi tipe set->selection;
        tentukan permulaan dan panjang string untuk matching;
        if(owner)
        begin
            letakkan pointer1 pada node terakhir list_owner;
            letakkan pointer2 pada node ke-1 list_member;
            while(!end list_member)
            begin
                if(pointer1->data==pointer2->data)
                begin
                    periksa informasi tipe set->order;
                    update pointer1->indek_member;
                    update pointer2->indek_owner;
                end
                letakkan pointer2 node next list_member;
            end while
        end
        if(member)
        begin
            letakkan pointer1 pada node terakhir list_member;
            letakkan pointer2 pada node ke-1 list_owner;
```

```
            while(!end list_owner)
                if(pointer1->data==pointer2->data)
                begin
                    periksa informasi tipe set->order;
                    update pointer1->indek_owner;
                    update pointer2->indek_member;
                end
                letakkan pointer2 pada node next list_owner;
            end while
        end
    end
    Update_list    _current_indikator_tipe_record_tipe_set_kontrol_unit    (alamat
    node);
end prosedur
```

3.2.10 Diskripsi fungsi pada proses Erase

Proses erase adalah proses menghapus sebuah record member dari database dengan memperhatikan *contraint retention.*

Prosedur utama yang terdapat pada proses erase adalah:

-prosedur *Hapus_node_pada_link_list* (nama tipe record) adalah prosedur yang menghapus node yang ditunjukkan *current* indikator kontrol unit. Untuk mengetahuinya dilakukan *prosedur Cari_pada_current_indikator kontrol_unit*(nama tipe record).

-prosedur Catat_informasi_tipe_set_yang_diload(nama_tipe_record) adalah prosedur yang menentukan apakah sebuah record dapat dihapus atau tidak berdasarkan informasi retention pada tipe set tersebut.

Algoritma prosedur diatas adalah :

```
prosedur Hapus_node_list(nama tipe record)
begin
    cari addres_node pada list current indikator kontrol unit;
    cari record tersebut menjadi owner pada set apa saja di indek_member;
```

periksa informasi tipe set->retention setiap set pada Catat_informasi_tipe set_yang_diload;
if(optional)
putus relasi dengan update indek_member;
if(fixed)
delete record owner dan member;
if(mandatory)
batalkan perintah Erase;
Update_list _current_indikator_tipe_record_tipe_set_kontrol_unit (alamat node);
end prosedur

3.2.11 Diskripsi fungsi pada proses Connect

Proses *connect* adalah proses membentuk relasi antara record pada kontrol unit dengan *set occurence* pada *list current* indikator tipe set dengan memperhatikan informasi tipe set seleksi dan *order* set. Prosedur utama dalam proses *connect* adalah :

-prosedur Connect_kan adalah prosedur yang melakukan pembentukan relasi antara record pada kontrol unit dengan *set occurence* pada current *indikator* tipe set. Untuk mendapatkan record pada kontrol unit dilakukan prosedur Cari_pada_current_indikator_kontrol_unit(nama tipe record). Sedangkan prosedur Cari_pada_list_current_indikator_tipe_set(nama tipe set) untuk mengetahui set occurence sebagai tujuan.

-prosedur Catat_informasi_tipe_set_yang_diload adalah prosedur untuk mengetahui informasi syarat membentuk relasi dan aturan urutan record member dalam set occurence.

Algoritma prosedur utama diatas adalah :

```
prosedur Connecting(nama tipe record,nama tipe set)
begin
    cari addres record yang akan diconnect di list current indikator kontrol unit;
    Cari_pada_current_indikator_kontrol_unit;
    pastikan bahwa record tersebut belum dihubungkan dalam set sesuai nama_set
    dengan mengecek indek_owner dan indek_member;
    if(tidak_terhubung)
    begin
        lihat informasi tipe set->insertion dan informasi tipe set->retention;
        if(manual)||(automatic optional)
        begin
            cari addres record yang akan disisipi di Set_current_set;
            tentukan apakah record tujuan tersebut owner atau member;
            lihat informasi tipe set->order;
            if(owner)
                update indek_member untuk disisipi;
            if(member)
            begin
                cari ownernya dengan mengecek di indek_owner record tersebut;
                update indek_member owner tersebut;
            end
        end
        Update_list _current_indikator_tipe_record_tipe_set_kontrol_unit (alamat
        node);
    end
end prosedur
```

3.2.12 Diskripsi fungsi pada proses Disconnect

Proses Disconnect adalah proses kebalikan dari proses connect yaitu memutuskan relasi antara record pada kontrol unit dengan *set occurence* pada list *current* indikator tipe set.

Algoritma prosedur utama pada proses disconnect adalah :

```
prosedur Disconnect_kan(nama tipe record,nama tipe set)
```

```
begin
    cari addres record yang akan didisconnect di  list_current_indikator_
    kontrol_unit;
    pastikan bahwa record tersebut dihubungkan dalam set sesuai nama_set dengan
    mengecek indek_owner dan indek_member;
    if(terhubung)
    begin
        lihat informasi tipe set->retention;
        if(optional)
        begin
            cari addres record yang akan didisconnect di
            list_current_indikator_tipe_set;
            tentukan apakah record tujuan tersebut owner atau member;
            if(owner)
                update indek_member untuk menghapus addres record yang ingin
                didisconnectkan;
            if(member)
            begin
                cari ownernya dengan mengecek di indek_owner record tersebut;
                update indek_member owner tersebut untuk menghapus addres
                record yang ingin disdisconnectkan;
            end
        end
        Update_list  _current_indikator_tipe_record_tipe_set_kontrol_unit  (alamat
        node);
    end
end prosedur
```

3.2.13 Diskripsi fungsi pada proses Reconnect

Proses reconnect adalah kombinasi prosedur pada proses *disconnect* dan *connect*.

Algoritma proses reconnect adalah :

```
prosedur Disconnect_lalu_connect(nama_record,nama_set)
begin
        lakukan prosedur seperti Disconnect  diteruskan prosedur Connect;
end prosedur
```

BAB IV

PEMBUATAN PERANGKAT LUNAK

4.1. Umum

Bab ini akan membahas pembuatan perangkat lunak dalam bahasa pemrograman Turbo C 2.0 dan untuk memanipulasi database digunakan Codebase 5.0. Program ini dibuat dalam beberapa fungsi dan prosedur yang melakukan proses tertentu. Dua proses utama yang dilakukan perangkat lunak ini adalah membangun model data *network* dan melaksanakan manipulasi model data *network*.

4.2. STRUKTUR DATA

Ada beberapa macam struktur data yang terdapat dalam program ini yaitu:

-Struktur data untuk menyimpan informasi *contraint* pada tipe set .

```
typedef struct relasi {
        int flag;
        char nama_set_type[16];
        char owner[11];       /* nama tipe record sebagai owner */
        char key_owner[26];
        char order[33];       /* aturan urutan member dalam set occurence */
        char member[11];  /* nama tipe record sebagai member */
        char key_member[26];
        char insertion[13]; /* aturan penyisipan member */
        char retention[15]; /* aturan keberadaan member dalam database */
```

```
        char selection[100];
} set[10];
```

-Struktur data untuk menyimpan data dari record *owner* dan record *member*.

```
typedef struct node {
        LINK4 link;
        char nama_set_type[16];
        char nama_rec[11];
        char *data;
        struct indek_member *indek_mem;  /* indek pointer record yang jadi
                                            membernya */
        struct  indek_owner *indek_owr;  /* indek pointer record  yang jadi
                                            ownernya */
} NODE;
```

-Struktur data untuk menyimpan indek penunjuk alamat dari suatu record ke record *owner*nya pada tipe set tertentu.

```
typedef struct indek_owner {
        char nama_set_type[16];
        struct node *owr;
        struct indek_owner *prev_or;
        struct indek_owner *next_or;
}INDEK_OR;
```

-Struktur data untuk menyimpan indek penunjuk alamat dari suatu record ke record-record *member* pada tipe set tertentu.

```
typedef  indek_member {
        char nama_set_type[16];
```

```
        struct membernya *alamat_first;
        struct indek_member *prev_mem;
        struct indek_member *next_mem;
} INDEK_MEM;
typedef struct membernya {
        char nama_set_type[16];
        struct node *mem;
        struct membernya *prev_member;
        struct membernya *next_member;
}MEMBERNYA;
```

-Struktur data untuk menyimpan penunjuk alamat record pada *current record indikator*.

```
typedef struct currensi_record {
        char *nama_record;
        struct node *alamat_rec;
        struct currensi_record *prev_currec;
        struct currensi_record *next_currec;
}CURREN_REC;
```

-Struktur data untuk menyimpan penunjuk alamat record pada *current set indikator.*

```
typedef struct currensi_set{
        char nama_set[16];
        struct node *alamat_rec;
        struct currensi_set *prev_curset;
        struct currensi_set *next_curset;
}CURREN_SET;
```

-Struktur data untuk menyimpan penunjuk alamat record pad *current unit indikator.*

```
typedef currensi_unit {
```

```
        char nama_record[11];
        struct node *alamat_unit;
}CURREN_UNIT;
```

-Variabel pointer

```
extern CURREN_REC  *awal_curr_rec;
extern CURREN_SET   *awal_curr_set;
extern CURREN_UNIT  *awal_curr_unit;
extern int DB_STATUS;
```

4.3. Membangun Model Data *Network*

Proses pembentukan ini dibagi atas tiga langkah utama yaitu *inisialisasi constraint*, membuat *linked list* , membangun relasi dengan memperhatikan aturan seleksi set dan aturan *order* set.

4.3.1. Inisialisi *Constraint*

Informasi *constraint* ini dipakai sebagai peraturan dalam proses pembentukan dan manipulasi model data *network*. Informasi ini disimpan dalam file *SKEMA.DBF*.

```
extern DATA4  *data_file;
extern FIELD4  *field_ref;
extern  CODE 4  *codebase;

int Open_skema()
{
    int  rc, j, jum_fields, in;
    char *isi_field;
    Open_file("SKEMA");
    jum_fields=d4num_fields(data_file);
    isi_field=(char*)malloc(100*sizeof(char));
```

```
for(rc=d4top(data_file);rc==r4success;rc=d4skip(data_file,1L), i++) {
    for(j=1;j<jum_fields;j++) {
        field_ref=d4field_j(data_file, j);
        strcpy(isi_field," ");
        isi_field=f4memo_str(field_ref);
        switch(j) {
            case 1: {
                strcpy(set[i].nama_set_type,isi_field);break;
            }
            case 2:{
                strcpy(set[i].owner,isi_field);break;
            }
            case 3:{
                strcpy(set[i].key_owner,isi_field);break;
            }
            case 4:{
                strcpy(set[i].order,isi_field);break;
            }
            case 5:{
                strcpy(set[i].member,isi_field);break;
            }
            case  6:{
                strcpy(set[i].key_member,isi_field);break;
            }
            case 7: {
                strcpy(set[i].insertion, isi_field);break;
            }
            case 8:{
                strcpy(set[i].retention, isi_field);break;
            }
            case 9:{
```

```
                strcpy(set[i].selection, isi_field);break;
            }
        }
    }
}
```

4.3.2. Membuat *Linked List.*

Dalam proses ini dilakukan penyimpanan data record dari tipe record *owner* dan *member* pada node-node *linked list*. Yang perlu dilakukan dalam proses ini adalah pencegahan terjadinya duplikasi. Hal ini karena pada model data *network* sebuah record *member* tidak boleh menjadi *member* lebih dari satu *set occurence* dalam sebuah tipe set .

```
NODE *Tambah_node(LIST4 *list,char *data,char nama_file[30])
{
    NODE *node_ptr;
    static char *temp;
    int i,duplicate;

    duplicate=0;
    temp=(char*)malloc(100*sizeof(char));
    strcpy(temp,data);
    node_ptr=(NODE*)l4first(list);
    while(node_ptr!=NULL) {
        node_ptr->data+=mulaikode1;
        data+=mulaikode1;
        if(strncmp(node_ptr->data,data,lengkode1)==NULL) {
            duplicate=100;
```

```
        }
        node_ptr->data-=mulaikode1;
        data-=mulaikode1;
        if((lengkode2!=0)&&(duplicate==100)) {
                node_ptr->data+=mulaikode2;
                data+=mulaikode2;
         if(strncmp(node_ptr->data,data,lengkode2)==NULL) {
                        duplicate=100;
                }
                else
                        duplicate=0;
               node_ptr->data-=mulaikode2;
                data-=mulaikode2;
        }
        if(duplicate!=100)
                node_ptr=(NODE*)l4next(list,node_ptr);
        else break;
}
if(duplicate==0) {
        node_ptr=(NODE*)u4alloc(sizeof(NODE));
        strcpy(node_ptr->nama_rec,nama_file);
        node_ptr->data=temp;
        node_ptr->indek_mem=NULL;
        node_ptr->indek_or=NULL;
        l4add(list,node_ptr);
        return(node_ptr);
}
else{
        Pesan("TERJADI DUPLICATE");getch();
        Delete(nama_file,data);
}
```

}

4.3.3. Membangun Relasi.

Proses ini diawali dengan menentukan syarat relasi antara record *owner* dan *member*. Syarat relasi ini diperoleh pada *constraint set selection*. Selanjutnya bila terpenuhi syarat relasi , urutan record-record *member* ditentukan *constraint set order*.

4.4. Manipulasi Model Data *Network*

Setiap proses manipulasi dilakukan pada awalnya dengan meneliti apakah setiap perintah manipulasi sesuai dengan format yang ditentukan. Selanjutnya dilakukan salah satu proses yaitu *navigasi, retrieval* dan memperbarui model data *network*.

Proses *navigasi* dilakukan untuk menentukan *current indikator* tipe record, tipe set atau *control unit* sehingga menunjuk kepada record dan *set occurence* tertentu dalam database. Setelah proses manipulasi dapat diketahui partisipasi record pada semua tipe set. Proses yang dilakukan untuk menentukan record *member* ke-satu, record *member* terakhir, record *member* setelah atau sebelum record *member* paling akhir dimanipulasi pada suatu *set occurence* adalah memeriksa record yang terakhir dimanipulasi pada *set occurence* tersebut berpartisipasi sebagai *owner* atau *member*. Bila sebagai *owner* proses pencariannya langsung pada *indek member* record tersebut dan bila sebagai *member* dilakukan pencarian record *owner*nya terlebih dahulu.

Proses *retrieval* dilakukan dengan mengambil record pada *current* indikator kontrol unit . Record pada model data *network* yang dapat dimanipulasi adalah record pada *current* indikator control unit. Tetapi sebelum record tersebut dapat dimanipulasi harus dilakukan proses ini.

Proses memperbaharui model data *network* dibagi dua yaitu memperbaharui record dan memperbarui tipe set. Memperbaharui record adalah proses menyimpan record, menghapus record yang tidak diinginkan dan merubah nilai suatu field dari record. Memperbaharui set adalah melakukan *connect* atau *disconnect* record *member* dalam sebuah *set occurence*, memindahkan record *member* dari *set occurence* menuju *set occurence* yang lain.

Setiap selesai melakukan manipulasi yang benar harus memperbaharui indikator baik tipe record, tipe set dan kontrol unit.

```
void Set_current_record(NODE *c)    /* c adalah pointer dari record yang baru
diakses */
{
    static CURREN_REC *d,*akhir;
    int sudah;

    sudah=0;
    d=awal_curr_rec;
    if(d!=NULL) {
       while(d!=NULL) {
          if(strcmp(d->nama_rec,c->nama_rec)==NULL) {
             if(strcmp(d->alamat_rec->data,c->data)!=NULL)
                d->alamat_rec=c;sudah=1;
          }
          if(d->next_currec==NULL) akhir=d;
```

```
            d=d->next_currec;
        }
        if(sudah==0) {   /* jika tipe record tersebut belum pernah diakses */
            d=(CURREN_REC*)u4alloc(sizeof(CURREN_REC));
            d->next_currec=NULL;
            strcpy(d->nama_rec,c->nama_rec);
            d->alamat_rec=c;
            akhir->next_currec=d;
            d->prev_currec=akhir;
        }
    }
    else {
        awal_curr_rec=(CURREN_REC*)u4alloc(sizeof(CURREN_REC));
        awal_curr_rec->next_currec=NULL;
        awal_curr_rec->prev_currec=awal_curr_rec;
        strcpy(awal_curr_rec->nama_rec,c22->nama_rec);
        awal_curr_rec->alamat_rec=c22;
    }
}
```

BAB V

UJI COBA DAN EVALUASI PERANGKAT LUNAK

5.1. Uji Coba

Uji coba perangkat lunak dilakukan untuk mengetahui hal-hal yaitu :

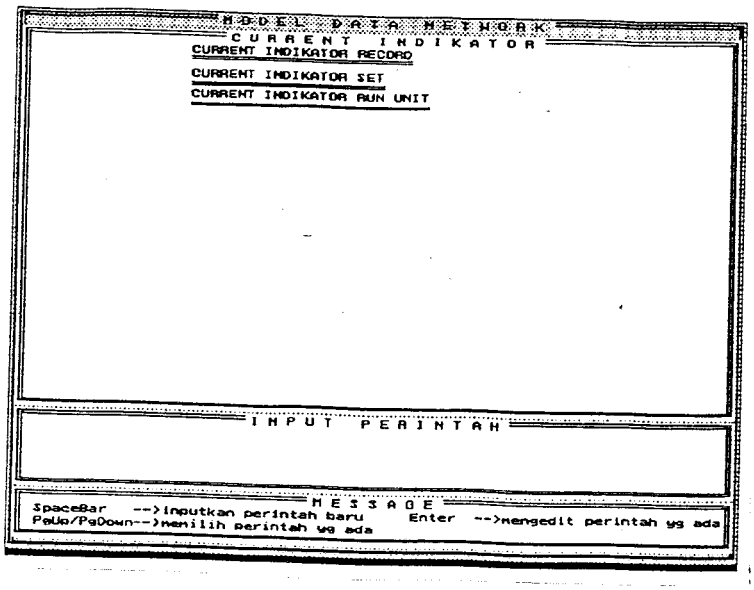
-Apakah tujuan dari pembuatan perangkat lunak terpenuhi dengan memperhatikan *constraint* model data *network* yaitu *insertion, retention,order dan set selection* . Kriteria yang digunakan digunakan yaitu memperhatikan *output* hasil pelaksanaan setiap perintah dan *currency indikator* tipe record, tipe set serta *control unit.*

-Kelebihan dan kekurangan perangkat lunak yang dapat diamati. Kelemahan yang ada akan dianalisa dan mungkin antisipasi yang dibutuhkan.

Ada tiga bagian utama yang ditampilkan oleh perangkat lunak seperti gambar 5.1 yaitu :

1. Bagian CURRENT INDIKATOR yaitu bagian di mana akan ditampilkan hasil perubahan *current* indikator tipe record, tipe set dan kontrol unit.
2. Bagian INPUT PERINTAH yaitu bagian untuk memasukkan perintah manipulasi.
3. Bagian MESSAGE yaitu bagian yang akan menampilkan pesan petunjuk atau kesalahan selama manipulasi.

Ketiga bagian utama itu seperti gambar 5.1



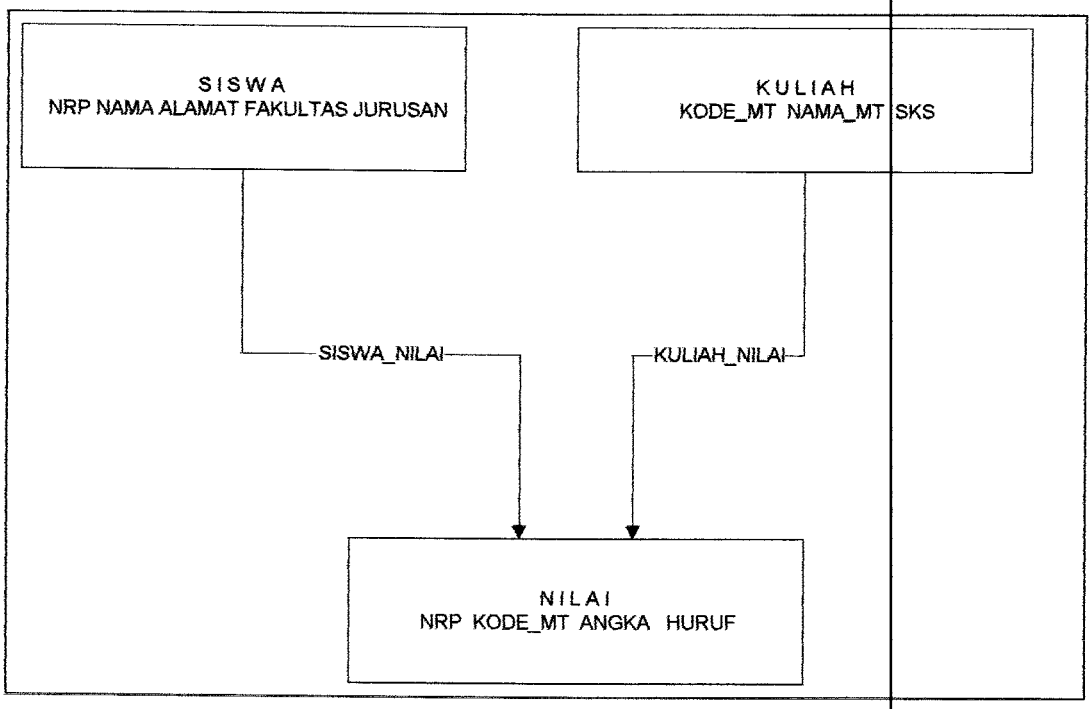
Gambar 5.1 Tampilan utama

Perintah untuk manipulasi pada perangkat lunak ini dapat dilakukan dengan dua cara yaitu :

a. Cara ke-satu adalah mengetikkan perintah langsung pada bagian Input Perintah dari perangkat lunak. Dengan cara ini setiap selesai pemasukkan satu perintah, perangkat lunak langsung mengeksekusi dan menampilkan perubahan pada current indikator tipe record, tipe set dan kontrol unit.

b. Cara ke-dua adalah memanfaatkan fasilitas EDIT pada DOS untuk mengetikkan beberapa perintah sekaligus dan menyimpan dalam file teks. Pada cara kedua ini dapat dimasukkan uji kondisi IF dan looping WHILE. Kemudian untuk mengeksekusinya perlu perintah DO nama file teks pada bagian Input Perintah.

Untuk uji coba ini telah disiapkan dua tipe set yaitu SISWA_NILAI dan KULIAH_NILAI. Gambar 5.2 menunjukkan diagram Bachman dari kedua tipe set tersebut

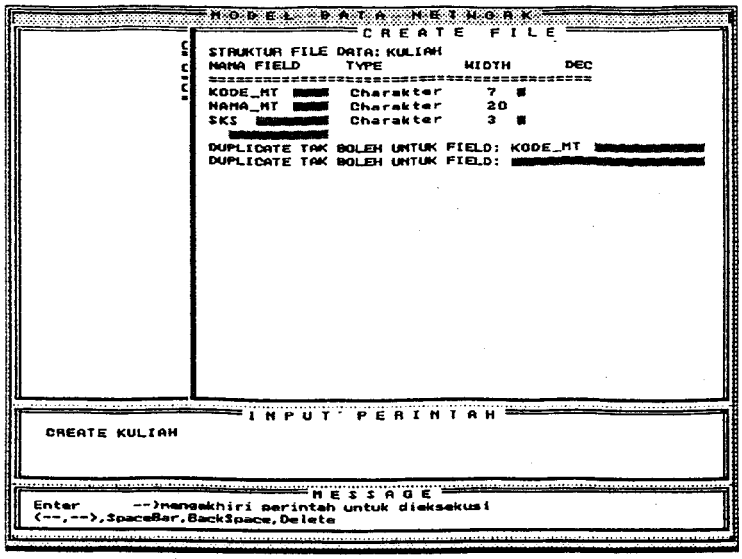


Gambar 5.2 Diagram Bahman

5.1.1 Uji Coba Cara Ke-satu

5.1.1.1 Deklarasi Tipe Record

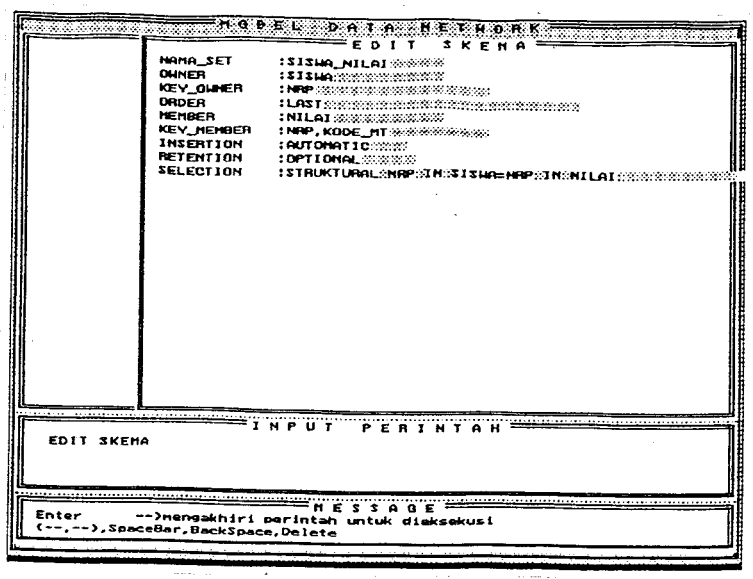
Untuk mendeklarasikan sebuah tipe record diketikkan CREATE nama_tipe_record pada bagian INPUT PERINTAH. Kemudian akan tampil fasilitas untuk mendeklarasi nama , tipe, panjang field serta deklarasi nilai field yang tidak boleh terjadi penggandaan. Uji coba ini dapat dilihat seperti gambar 5.3.



Gambar 5.3 Uji coba deklarasi tipe record

5.1.1.2 Deklarasi Tipe Set

Untuk mendeklarasikan sebuah tipe set caranya seperti deklarasi tipe record tetapi nama_tipe_record harus SKEMA dan nama field tertentu yaitu nama_tipe_set, owner, key_owner, order, member, key_member, insertion, retention, seleksi. Untuk mengedit deklarasi tipe set yang sudah ada dilakukan perintah EDIT SKEMA sehingga nilai tiap field dapat dirubah misalnya nilai field order dapat dipilih FIRST, LAST, PRIOR, NEXT atau SORTED. Kemudian nilai field insertion dapat dipilih AUTOMATIC atau MANUAL. Nilai field retention dapat dipilih OPTIONAL, MANDATORY atau FIXED. Nilai seleksi dapat dipilih STRUKTURAL atau APPLIKASI. Uji coba ini seperti gambar 5.4.



Gambar 5.4 Uji coba perintah EDIT SKEMA

5.1.1.3 Perintah Build_network

Uji coba ini dilakukan dengan mengetikkan perintah LOAD nama_tipe_set pada bagian INPUT PERINTAH kemudian perangkat lunak akan membentuk relasi seperti deklarasi tipe set tersebut pada SKEMA misalnya LOAD SISWA_NILAI, LOAD KULIAH_NILAI.

5.1.1.4 Perintah Navigasi

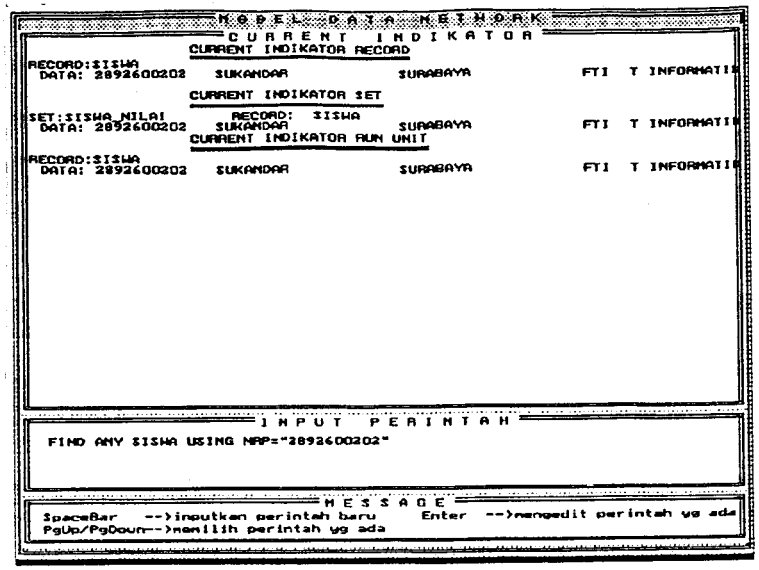
Uji coba ini dilakukan dengan mengetikkan perintah yang dimulai dengan FIND dengan aturan format seperti pada BAB II. Pada setiap akhir eksekusi dapat dilihat perubahan *current* indikator tipe record, tipe set dan kontrol unit pada bagian CURRENT INDIKATOR perangkat lunak. Contoh uji coba yang dilakukan adalah sebagai berikut :

LOAD SISWA_NILAI Memanggil tipe set SISWA_NILAI

LOAD KULIAH_NILAI Memanggil tipe set KULIAH_NILAI

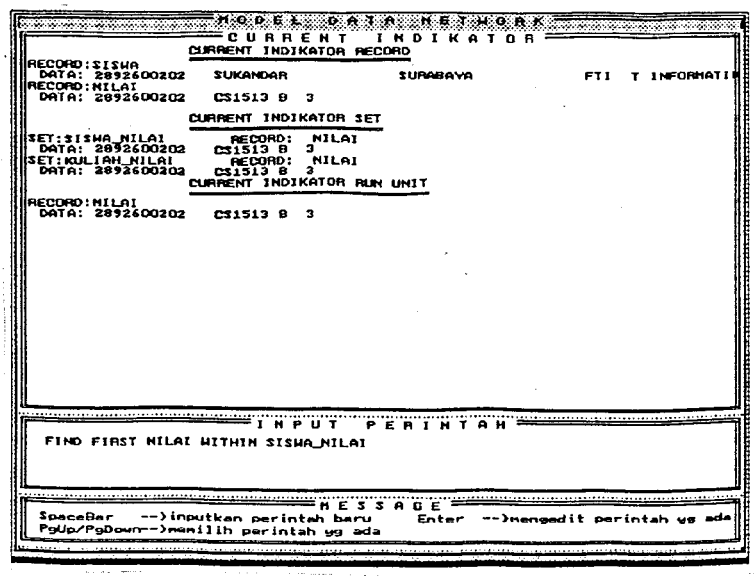
FIND ANY SISWA USING NRP="2892600202" Mencari record SISWA yang mempunyai nilai NRP="2892600202"

Hasil uji coba dapat diperhatikan pada bagian CURRENT INDIKATOR. Untuk *current* indikator tipe record SISWA, tipe set SISWA_NILAI , kontrol unit menunjukkan record yang mempunyai NRP="2892600202". Hal ini menunjukkan bahwa record tersebut berpartisipasi hanya pada tipe set SISWA_NILAI. Uji coba ini dapat dilihat pada gambar 5.5



Gambar 5.5 Uji coba FIND ANY

Setelah uji coba diatas dapat dilanjutkan dengan perintah navigasi yang lain misalnya FIND FIRST NILAI WITHIN SISWA_NILAI diketikkan pada bagian INPUT PERINTAH. Tujuan perintah ini adalah mencari record *member* NILAI yang ke-satu dalam *set occurence* yang ditunjukkan pada *current* indikator tipe set SISWA_NILAI. Hasil uji coba dapat dilihat pada gambar 5.6.

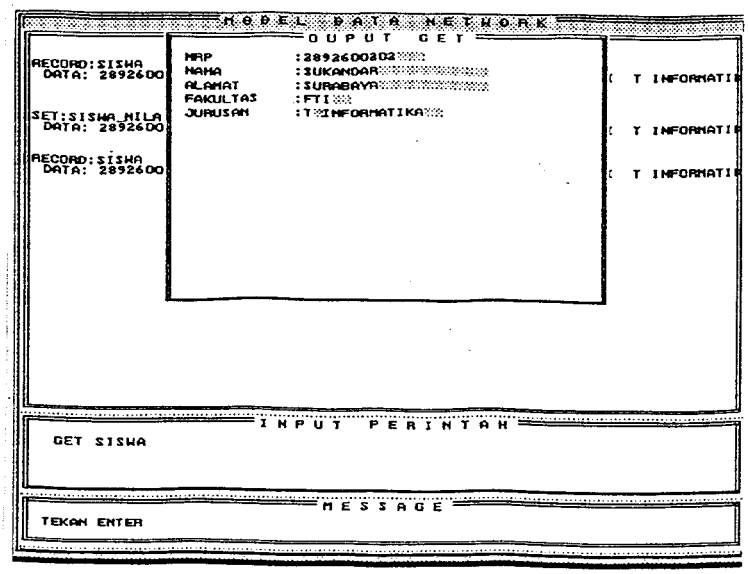


Gambar 5.6 Uji coba FIND FIRST

Perubahan terjadi pada *current* indikator, sekarang tipe record bertambah tipe record NILAI, tipe set bertambah KULIAH_NILAI ini menunjukkan record NILAI tersebut berpartisipasi pada tipe set SISWA_NILAI dan KULIAH_NILAI, kontrol unit menunjuk pada record NILAI yang dicari.

5.1.1.5 Perintah Retrieval

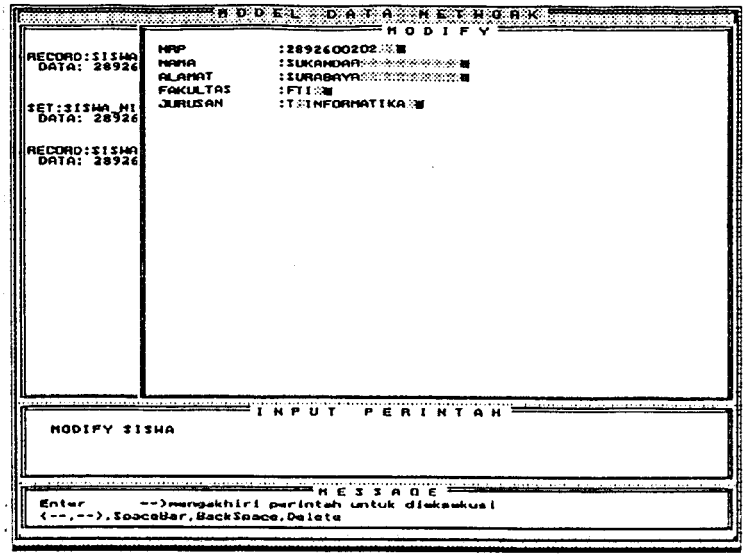
Uji coba perintah *retrieval* adalah dengan mengetikkan perintah GET nama tipe record yang terdapat pada kontrol unit. Contoh uji coba adalah setelah perintah FIND ANY SISWA USING NRP="2892600202" diatas lalu dilakukan GET SISWA. Hasil uji coba seperti gambar 5.7. Perintah ini bertujuan menyalin nilai record pada kontrol unit dalam *user work area*(UWA) sehingga dapat dimanipulasi.



Gambar 5.7 Uji coba perintah Get

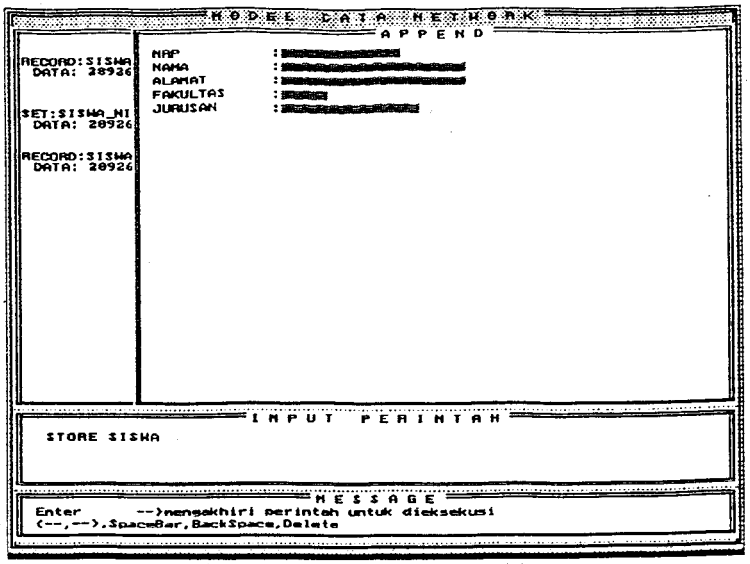
5.1.1.6 Perintah Update

Uji coba perintah update dapat dilakukan pada sebuah record atau pada sebuah set occurence. Uji coba pada tipe record yang terdapat pada kontrol unit dilakukan dengan mengetikkan perintah MODIFY nama tipe record seperti gambar 5.8 dimana pada kontrol unit ada tipe record SISWA.



Gambar 5.8 Uji coba perintah MODIFY

Uji coba untuk menambah record pada database dan membentuk relasi dilakukan dengan perintah STORE seperti gambar 5.9



Gambar 5.9 Uji coba perintah STORE

Uji coba *update* set *occurence* tipe set pada *current* indikator tipe set dilakukan dengan DISCONNECT, CONNECT atau RECONNECT. Uji coba yang dilakukan menunjukkan hasil yang sesuai diharapkan teori manipulasi model data *network.*

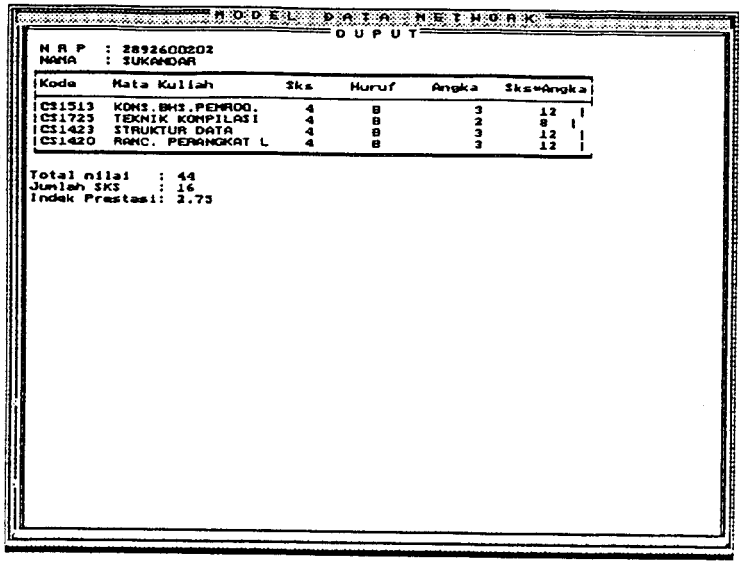
5.1.2 Uji Coba Cara Ke-dua

Uji coba dilakukan dengan memanfaatkan fasilitas EDIT pada DOS untuk mengetikkan contoh program. Misalnya program yang disimpan dalam TRANS.TXT. Dibawah ini program utama dari TRANS.TXT.

```
load siswa_nilai;      { memanggil tipe set siswa_nilai}
load kuliah_nilai;     { memanggil tipe set kuliah_nilai}
find any siswa using nrp="2892600202";
get siswa;         { menyalin record SISWA kontrol unit ke UWA}
println (" NRP     :");  {mencetak string NRP }
print siswa.nrp;         { mencetak nilai SISWA.NRP dari UWA }
println (" NAMA:");
```

```
print siswa.nama;
jum_sks=0;
total=0;
while DB_STATUS
{
        find next nilai within siswa_nilai;  {mencari record member NILAI dari set
                                              occurence current indikator tipe set }
        get nilai;
        nilai_angka=atoi(nilai.angka);
        nilai_huruf=nilai.huruf;
        find owner within kuliah_nilai; { mencari owner dari record NILAI }
        get kuliah;
        s_k_s=atoi(kuliah.sks);
        jum_sks=jum_sks+sks;
        sks_angka=s_k_s*nilai_angka;
        total=total+sks_angka;
        print nilai_huruf;
        print nilai_angka;
        print sks_angka;
}
ip=total/jum_sks;
print total;
print jum_sks;
print ip;
```


Hasil uji coba terlihat pada gambar 5.10.



Gambar 5.10 Uji coba TRANS.TXT

5.2. Evaluasi Prosedur

Dalam implementasi prosedur ada beberapa hal yang perlu diperhatikan yaitu :

-Proses pencegahan record dengan nilai *field* kunci yang lebih dari satu sangat penting karena model data *network* membentuk relasi *one to many* antara tipe record *owner* dan tipe record *member*.

-Perangkat lunak menerapkan *insertion constraint* untuk menentukan prosedur penyisipan record *member* pada *set occurence*, *retention constraint* untuk menentukan keberadaan suatu record dalam database, *order constraint* untuk membentuk urutan record member.

-Perangkat lunak dapat menunjukkan partisipasi tipe record pada seluruh tipe set sehingga memudahkan mengolah database yang besar.

BAB VI

KESIMPULAN DAN SARAN

6.1. Kesimpulan

Dari uji coba dan dasar teori manipulasi model data *network* maka dapat disimpulkan hal-hal sebagai berikut :

a. Relasi yang dapat dilakukan dalam model data *network* adalah *one to one* dan *one to many* antara record *owner* dan *member*. Untuk mendukung ini dilakukan pencegahan terjadinya duplikasi sehingga sebuah record tidak boleh menjadi *member* lebih dari satu set *occurence* dalam sebuah tipe set. Untuk relasi *many to many*, model data *network* membuat dua tipe set. yang masing-masing menerapkan relasi *one to many* dan record *dummy atau interseksi*. Record ini terdiri dari *field* kunci dari record *owner* pada dua tipe set tersebut.

b. Dalam model data *network*, sebuah record dapat berpartisipasi baik sebagai *owner* dan *member* pada beberapa tipe set yang berbeda. Dengan demikian *entry point* untuk menelusuri dan memanipulasi dalam database model data *network* banyak.

c. Penentuan kombinasi jenis *insertion* dan *retention* pada *constraint* tipe set penting. Karena pemilihan kombinasi yang tidak tepat, pada akhirnya dalam memanipulasi model data *network* tidak bermanfaat.

d. Dalam memanipulasi model data *network*, peran *current indikator* untuk record, tipe set dan *control* unit sangat penting. Hal ini karena

manipulasi model data ini adalah *record at a time* yaitu dalam satu waktu proses manipulasi ditujukan hanya pada satu record saja. Record yang dapat dimanipulasi adalah record yang terdapat di *current indikator* kontrol unit.

6.2. SARAN

Dengan melihat kesimpulan di atas, maka pengembangan perangkat lunak untuk manipulasi banyak tipe set dengan tetap memperhatikan aturan model data network bermanfaat besar dalam memecahkan masalah rumit dalam database. Oleh karena itu penambahan jumlah tipe record dan tipe set dapat dilakukan melalui cara sebagai berikut:

- mendeklarasikan tipe record baru dan menambah data baru dengan perintah CREATE <nama_tipe_record> , APPEND atau STORE.
- menambah jumlah tipe set dengan perintah APPEND dan EDIT SKEMA.
- memperbaiki penanganan penunjuk alamat(*pointer*).

Dengan demikian manfaat dari perangkat lunak lebih besar lagi.

DAFTAR PUSTAKA

1. Borland, Turbo C Reference Guide , Borland Internastional Inc , USA ,1988.
2. Borland, Turbo C User's Guide , Borland International Inc , USA ,1988.
3. Henry F Korth Abraham Silber Schatz , Database System Concepts, McGraw-Hill , New York , 1986.
4. J.H. Ter Bekke, Semantic Data Modeling , First edition , Prentice Hall , Englewood Cliffs , New Jersey, 1992.
5. Ramez Elmasri and B. Navanthe , Fundamental Of Database System , The Benjamin / Cumming , 1989.
6. Roger S Pressman Ph.D , Software Engineering , Second edition , McGraw-Hill, New York , 1987.

Lampiran A
Petunjuk Menjalankan Perangkat Lunak

Pertama kali sebelum dijalankan konfigurasi sistem atau base memory harus maksimal dengan memperkecil jumlah file sistem yang diload waktu booting.

Langkah ke-1 : Memanggil perangkat lunak

Perangkat lunak ini dijalankan dengan memanggil file exenya pada prompt DOS diakhiri dengan menekan Enter.

A-> net

Untuk memasukkan perintah baru didahului dengan menekan SPACEBAR dan bila diinginkan mengulang perintah terdahulu dapat dilakukan dengan menekan PgUp atau PgDn untuk mencari perintah tersebut.

Langkah ke-2 : Memanggil tipe set

Bila diperlukan untuk merubah deklarasi tipe set dilakukan dengan mengetikkan perintah EDIT SKEMA pada bagian Input Perintah.

Kemudian setelah tipe set siap lalu dipanggil dengan mengetikkan perintah LOAD nama_tipe_set misalnya LOAD SISWA_NILAI. Bila diperlukan beberapa tipe set, nama tipe set dapat dipanggil sekaligus dengan perintah LOAD.

Langkah ke-3 : Mengeset current indikator

Selanjutnya harus diketikkan perintah navigasi FIND ANY untuk mengeset nilai current indikator. Kemudian dapat dilakukan perintah manipulasi yang lainnya.

Langkah ke-4 : Memasukkan perintah lainnya.

Setelah current indikator menunjuk pada nilai tertentu, maka perintah manipulasi yang lain dapat diketikkan sesuai tujuan yang diinginkan. Format perintah dan tujuan dapat dilihat pada BAB II.

Langkah ke-5 : Keluar dari perangkat lunak.

Untuk keluar dari perangkat lunak diketikkan QUIT diakhiri menekan Enter.

Pada langkah ke-2 sampai ke-5 dapat digabungkan dalam sebuah program manipulasi sederhana dengan memanfaatkan EDIT pada DOS. Kemudian untuk mengeksekusi diketikkan DO nama_program pada bagian Input Perintah, misalnya DO TRANS lalu ditekan Enter.